Self-managed Workflows for Cyber-physical Systems

Dissertation

to achieve the academic degree
Doktor-Ingenieur (Dr.-Ing.)

Submitted at

Technische Universität Dresden
Fakultät Informatik

Submitted by

Dipl.-Inf. Ronny Seiger
born 19.12.1985 in Cottbus

1st Referee
Prof. Dr.-Ing. Thomas Schlegel
(Technische Universität Dresden/Hochschule Karlsruhe, Deutschland)

2nd Referee
Prof. Dr. Mathias Weske
(Hasso-Plattner-Institut, Universität Potsdam, Deutschland)

Subject Consultant
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
(Technische Universität Dresden, Deutschland)

Co-Supervisor
Prof. Dr. rer. nat. habil. Uwe Aßmann
(Technische Universität Dresden, Deutschland)

Submitted on 17th July 2018
Defended on 7th November 2018

Confirmation

I hereby certify that I have authored this Dissertation entitled *Self-managed Workflows for Cyber-physical Systems* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the spiritual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, November 19, 2018

Dipl.-Inf. Ronny Seiger

Acknowledgements

"If all difficulties were known at the outset of a long journey, most of us would never start out at all."

Dan Rather

Undoubtedly, the journey of a PhD student wandering around in the wide fields of computer science is a long journey full of difficulties, obstacles and side quests that distract the player from the main story. During the course of the years, I had many opportunities to get insights into related research fields and communities, to travel the wasteland to conferences and meet people at interesting places, and to work with a variety of researchers, students and other NPCs on various projects. This helped me a lot to develop myself and my special perks as well as to level up my skills as a scientist. With making good progress in the main story, after 6,5 years and 6 research projects at two chairs, I was able to focus and advance my research to finally and gladly arrive at the end of this long journey on the difficult road towards the final quest of defeating the end-boss(es) and finishing the main PhD storyline. At this point, I would like to thank all people who supported me along the road towards this final level of my thesis.

First of all, I would like to thank my supervisors Prof. Thomas Schlegel and Prof. Uwe Aßmann for giving me the opportunity to work in this very vibrant and interesting research field. They always provided me with valuable feedback and new ideas to extend and improve the current state of the PhD project and thesis. In addition, I would like to thank Prof. Schill and Prof. Weske and their research groups for their co-supervision and additional feedback.

Many special thanks go to my current and former colleagues from the Software Technology group and Software Engineering of Ubiquitous Systems group. First and foremost, I would like to mention and thank Steffen Huber, Christoph Seidl, Peter Heisig, Florian Niebling and Christine Keller for the intense discussions of ideas and concepts, which led to the development of this PhD thesis. Of course, my thanks also go to all the other colleagues and NPCs who have to endure my madness every day. Besides discussions about work-related stuff, we always had good off-topic chats about the meaning of life and all the rest, sometimes over a beer or two. Additionally, I am grateful for the support from my colleagues from N+P Informationssyteme GmbH in Meerane, from the Institute of Ubiquitous Mobility Systems in Karlsruhe, and from Berufsakademie Dresden. Last but not least, countless paper reviewers and discussions with associated researchers at various venues helped me to mature the concepts presented in this thesis to their present state.

I would also like to extend my gratitude to the countless students working hard and getting their hands dirty to make sense of my fuzzy ideas and implement parts of the prototypes presented in this thesis. Special thanks with that regard go to André

Kühnert, Stefan Herrmann, Reik Müller and all other of my former and current student assistants, thesis students as well as seminar and practical course students.

Most importantly, I would like to thank my family and friends for supporting and believing in me the whole time: my parents for giving me the opportunity to move to the big city to study computer science and supporting me during the sometimes very hard and difficult times with their love (and also some bottlecaps from time to time to survive in the wasteland); to my dearest Bets for her love and warmth and understanding during the last months of writing; and of course to all my friends for providing badly needed distractions from work and thesis writing–be it floating on the Soča river while listening to our favourite metal bands, driving for hours through Cambodia's countless temple areas in an uncomfortable Tuk-Tuk, or hunting kangaroos with Staubi along the coast of Western Australia. All in all, being a PhD student and research assistant was an awesome experience and I had a great time working together with colleagues, students, supervisors and other researchers. I hope, I will be able to continue my work in an academic or R&D-related context in the future.

Ronny

This research has received funding under the grant numbers 100098171 ("VICCI" project) and 100268299 ("CyPhyMan" project) by the European Social Fund (ESF) and the German Federal State of Saxony.

Abstract

Workflows are a well-established concept for describing business logics and processes in web-based applications and enterprise application integration scenarios on an abstract implementation-agnostic level. Applying Business Process Management (BPM) technologies to increase autonomy and automate sequences of activities in Cyber-physical Systems (CPS) promises various advantages including a higher flexibility and simplified programming, a more efficient resource usage, and an easier integration and orchestration of CPS devices. However, traditional BPM notations and engines have not been designed to be used in the context of CPS, which raises new research questions occurring with the close coupling of the virtual and physical worlds. Among these challenges are the interaction with complex compounds of heterogeneous sensors, actuators, things and humans; the detection and handling of errors in the physical world; and the synchronization of the cyber-physical process execution models. Novel factors related to the interaction with the physical world including real world obstacles, inconsistencies and inaccuracies may jeopardize the successful execution of workflows in CPS and may lead to unanticipated situations.

This thesis investigates properties and requirements of CPS relevant for the introduction of BPM technologies into cyber-physical domains. We discuss existing BPM systems and related work regarding the integration of sensors and actuators into workflows, the development of a Workflow Management System (WfMS) for CPS, and the synchronization of the virtual and physical process execution as part of self-* capabilities for WfMSes. Based on the identified research gap, we present concepts and prototypes regarding the development of a CPS WFMS w. r. t. all phases of the BPM lifecycle. First, we introduce a CPS workflow notation that supports the modelling of the interaction of complex sensors, actuators, humans, dynamic services and WfMSes on the business process level. In addition, the effects of the workflow execution can be specified in the form of goals defining success and error criteria for the execution of individual process steps. Along with that, we introduce the notion of *Cyber-physical Consistency*. Following, we present a system architecture for a corresponding WfMS (*PROtEUS*) to execute the modelled processes–also in distributed execution settings and with a focus on interactive process management. Subsequently, the integration of a cyber-physical feedback loop to increase resilience of the process execution at runtime is discussed. Within this MAPE-K loop, sensor and context data are related to the effects of the process execution, deviations from expected behaviour are detected, and compensations are planned and executed. The execution of this feedback loop can be scaled depending on the required level of precision and consistency. Our implementation of the MAPE-K loop proves to be a general framework for adding self-* capabilities to WfMSes. The evaluation of our concepts within a smart home case study shows expected behaviour, reasonable execution times, reduced error rates and high coverage of the identified requirements, which makes our CPS WfMS a suitable system for introducing workflows on top of systems, devices, things and applications of CPS.

Publications

This thesis is partially based on the following peer-reviewed publications:

- Thomas Schlegel, Krešimir Vidačković, Sebastian Dusch, and Ronny Seiger. Management of interactive business processes in decentralized service infrastructures through event processing. *Journal of King Saud University - Computer and Information Sciences*, 24(2):137 – 144, 2012

- Ronny Seiger, Christine Keller, Florian Niebling, and Thomas Schlegel. Modelling complex and flexible processes for smart cyber-physical environments. In *Proceedings of 25th European Modeling and Simulation Symposium (EMSS)*, pages 73–82, September 2013

- Ronny Seiger, Florian Niebling, and Thomas Schlegel. A distributed execution environment enabling resilient processes for ubiquitous systems. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 220–223, March 2014

- Ronny Seiger, Susann Struwe, Sandra Matthes, and Thomas Schlegel. A resilient interaction concept for process management on tabletops for cyber-physical systems. In *Human Interface and the Management of Information. Information and Knowledge in Applications and Services*, pages 347–358. Springer, 2014

- Ronny Seiger, Christine Keller, Florian Niebling, and Thomas Schlegel. Modelling complex and flexible processes for smart cyber-physical environments. *Journal of Computational Science*, 10:137 – 148, 2015

- Ronny Seiger. Modelling and execution of consistent and distributed workflows for cyber-physical systems. In *Business Process Management (Doctoral Consortium)*, 2015

- Ronny Seiger, Steffen Huber, and Thomas Schlegel. *PROtEUS: An Integrated System for Process Execution in Cyber-Physical Systems*, pages 265–280. Springer International Publishing, 2015

- Ronny Seiger, Christoph Seidl, Uwe Aßmann, and Thomas Schlegel. A capability-based framework for programming small domestic service robots. In *Proc. of the 2015 Joint MORSE/VAO Workshop.*, pages 49–54, New York, NY, USA, 2015. ACM

- Ronny Seiger, Steffen Huber, and Thomas Schlegel. Toward an execution system for self-healing workflows in cyber-physical systems. *Software & Systems Modeling*, pages 1–22, 2016

- Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. *Enabling Self-adaptive Workflows for Cyber-physical Systems*, pages 3–17. Springer International Publishing, 2016
- Ronny Seiger, Diana Lemme, Susann Struwe, and Thomas Schlegel. An interactive mobile control center for cyber-physical systems. In *Proc. of the Int. Joint Conf. on Pervasive and Ubiquitous Computing: Adjunct*, pages 193–196, New York, NY, USA, 2016. ACM
- Steffen Huber, Ronny Seiger, André Kühnert, Vasileios Theodorou, and Thomas Schlegel. Goal-based semantic queries for dynamic processes in the internet of things. *International Journal of Semantic Computing*, 10(02):269–293, 2016
- Steffen Huber, Ronny Seiger, André Kühnert, and Thomas Schlegel. Using semantic queries to enable dynamic service invocation for processes in the internet of things. In *IEEE Intern. Conference on Semantic Computing (ICSC)*, pages 214–221, Feb 2016
- Steffen Huber, Ronny Seiger, André Kühnert, and Thomas Schlegel. A context-adaptive workflow engine for humans, things and services. In *Proc. of the ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, pages 285–288, New York, NY, USA, 2016. ACM
- Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software & Systems Modeling*, Nov 2017
- Ronny Seiger, Stefan Herrmann, and Uwe Aßmann. Self-healing for distributed workflows in the internet of things. In *IEEE Int. Conference on Software Architecture (ICSA) Workshops*, 2017
- Ronny Seiger, Steffen Huber, and Thomas Schlegel. An execution system for self-healing workflows in cyber-physical systems. In *Software Engineering 2017*, number Lecture Notes in Informatics (LNI), pages 75–76. Gesellschaft für Informatik, 2017
- Ronny Seiger, Steffen Huber, and Peter Heisig. Proteus++: A self-managed iot workflow engine with dynamic service discovery. In *9th Central European Workshop on Services and their Composition (ZEUS)*, 2017
- Ronny Seiger, Mandy Korzetz, Maria Gohlke, and Uwe Aßmann. Mixed reality cyber-physical systems control and workflow composition. In *Proc. of the 16th Int. Conf. on Mobile and Ubiquitous Multimedia*, MUM '17. ACM, 2017
- Ronny Seiger, Stefen Huber, and Uwe Aßmann. A case study for workflow-based automation in the internet of things. In *IEEE Int. Conference on Software Architecture (ICSA) Companion*, 2018
- Ronny Seiger, Peter Heisig, and Uwe Aßmann. Retrofitting of workflow management systems with self-x capabilities for internet of things. In *BP-Meet-IoT Workshop, Int. Conference on Business Process Management (BPM) Workshops*, 2018

The following peer-reviewed publications cover work that is closely related to the content of the thesis, but not contained herein:

- Ronny Seiger, Stephan Groß, and Alexander Schill. Seccsie: A secure cloud storage integrator for enterprises. In *2011 IEEE 13th Conference on Commerce and Enterprise Computing*, pages 252–255, Sept 2011

- Georg Püschel, Ronny Seiger, and Thomas Schlegel. Test modeling for context-aware ubiquitous applications with feature petri nets. In *Proc. Workshop Model-based Interactive Ubiquitous Systems (MODIQUITOUS)*, 2012

- Ronny Seiger, Tobias Nicolai, and Thomas Schlegel. A framework for controlling robots via brain-computer interfaces. In Andreas Butz, Michael Koch, and Johann Schlichter, editors, *Mensch & Computer 2014 - Workshopband*, pages 003–006, Berlin, 2014. De Gruyter Oldenbourg

- Ronny Seiger, Florian Niebling, Mandy Korzetz, Tobias Nicolai, and Thomas Schlegel. A framework for rapid prototyping of multimodal interaction concepts. *Large-scale and Model-based Interactive Systems*, pages 21–28, 2015

- Thomas Schlegel, Ronny Seiger, Christine Keller, and Romina Kühn. Model-based interactive ubiquitous systems (modiquitous). In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 296–297, New York, NY, USA, 2015. ACM

- Ronny Seiger, Bashar Altakrouri, Andreas Schrader, and Thomas Schlegel, editors. *Proceedings of the 1st Workshop on Large-scale and Model-based Interactive Systems: Approaches and Challenges (LMIS 2015)*, number 1380 in CEUR Workshop Proceedings, Aachen, 2015

Contents

1. Introduction

> "A customer can have a car painted any color he wants as long as it's black."
>
> Henry Ford

1.1. Motivation

This famous quote by American entrepreneur Henry Ford represents one of the key statements of the Second Industrial Revolution. Henry Ford was one of the pioneers in the field of mass production in the beginning of the 20th century. His dream of manufacturing affordable automobiles for everyone led to the invention of the conveyor belt, the electrification of mechanical production processes and the emergence of mass production, which are important corner stones of the Second Industrial Revolution. The *Model T* first built by Ford in 1908 is one of the symbols of this era. However, Ford's quote also shows that production processes of complex and rather expensive goods were required to be highly inflexible and static in order to be feasible for mass market production.

Now, roughly a 100 years later everyone in the German production and software industries is talking about *Industry 4.0*, a term coined at the Hannover Fair in 2011 [KLW11]. After the introduction of advanced electronics and computers to control production processes–known as Third Industrial Revolution–Industry 4.0 aims at a stronger integration and tighter coupling of information technologies with machines, tools, material, produced goods, and also customers along the whole supply chain and during all phases of the product lifecycle (*Digital Twin*) [LFK+14, SCA+17, RvWLB15]. This currently ongoing digitisation eventually leads to the emergence of an *Internet of Everything* [DMLYE18], which will pervade every area of life. One of the goals of the Fourth Industrial Revolution is the establishment of highly flexible, decentralized production processes involving virtual and physical entities, which can be reconfigured instantaneously according to the current demand and various context factors, and which are feasible even for lot sizes of one item. Research in the areas of Cyber-physical Systems (CPS) and Internet of Things (IoT) will provide the technological foundations for achieving this goal [Jaz14].

Business Process Management (BPM) and workflow technologies have provided well-established concepts and approaches to flexibly orchestrate web applications based on Service-oriented Architectures (SOAs) and to realize the integration of existing systems and software in Enterprise Application Integration (EAI) contexts. Many organizations use business processes to formalize, execute and monitor their intra-organizational as well as inter-organizational processes. These technologies prove feasible to manage workflows regarding virtual resources and purely digital

services. One of the most common examples for introducing and explaining business processes is the booking of a business trip involving flight companies, hotels, rental car agencies and credit card companies [Whi05]. This scenario shows the strength of business processes when orchestrating web services and routing data among various workflow tasks within an organization or across multiple companies and enterprise applications involved in the processes.

With current developments in the field of micro-electronics, more and more hardware devices and physical objects (*Things*) are equipped with sensors, actuators and microprocessors, which allow data to be gathered from these devices and the devices to be controlled by software. Many of these IoT devices and their control software are not closed anymore, they increasingly offer open programming interfaces nowadays and therefore, the possibilities to use or add web services to control and connect them remotely. This raises the question about the suitability of workflow technologies to be used to automate processes in the context of CPS. The central goal of this thesis is to investigate this question as workflows would allow for an easy and flexible programming and orchestration of tasks and more complex processes among the sensors, actuators, embedded computers, desktop and mobile computers, as well as Cloud servers of CPS. A main advantage of using workflows in this context is the integration of functionality from different, highly heterogeneous devices and platforms on a unified abstraction layer in the sense of EAI, which also facilitates reuse of functionality as well as the simplified creation of high-level programs across system and device boundaries [SHS15]. Thus, the application of BPM technologies in CPS and IoT environments represents a new and vibrant research area (*BPM Everywhere* [Cha15]), which faces new challenges and requirements with the design of a Workflow Management System (WfMS) that is able to enact workflows in CPS [JKM+17, MBBF17]. Existing WfMSes from industry and research mostly put focus on aspects regarding digital business processes in organizational contexts. The extension of business processes into the physical domain and with that, the need for more adaptive and context-sensitive workflows that also consider the physical effects of the workflow execution are topics addressed by only a few related approaches.

This PhD thesis investigates the topic of workflows for CPS in depth. Based on the characteristics of CPS, a set of requirements for a WfMS operating in the context of CPS is derived. We design a workflow notation and management system for CPS, which take this set of new requirements into account. Special focus is put on the interaction of the processes with the physical world via sensors, actuators and humans, and on the implementation of a feedback loop, which considers data from additional sources to verify the process execution or to handle unanticipated errors that may have occurred–thus increasing the resilience of the WfMS through self-adaptation. As many existing BPM systems do not fulfil the necessary requirements to be capable of self-management in the context of CPS, we also propose a framework and retrofitting process to add this capability to other workflow systems. To illustrate and evaluate our new concepts, example processes from the *Smart Home* domain as application area of Industry 4.0 technologies are used throughout this thesis. Besides Smart Factories in the context of Industry 4.0, smart homes represent instances of CPS that consist of more or less complex sensor and actuator networks providing assistance to their residents. Like other smart spaces, these CPS are controlled by various context-aware applications and adaptive smart processes

(cf. Figure 1.1). However, compared to smart factories, smart homes are characterised by a higher degree of unanticipated behaviour and more frequent context changes as the residents–humans and their pets–tend to be very dynamic and less predictable than production machines, which leads to the constant emergence of new situations and possible errors that need to be anticipated. Therefore, processes have to be more flexible, adaptive and resilient against possible errors and newly emerging situations. Another advantage of smart homes as testbeds for cyber-physical workflows is the availability of a wide range of affordable smart consumer appliances that provide open programming and communication interfaces. Production machines used in industry are very costly and currently mostly controlled by closed proprietary software, which prevents runtime adaptations and the implementation of a workflow layer on top of these machines to orchestrate the production processes from a higher level. There is a high probability that this fact will change in the near future with further development of concepts and technologies for Industry 4.0. The findings from this thesis regarding workflows for CPS can then also be applied to these Cyber-physical Production Systems (CPPS) [Mon14] and other emerging smart spaces.

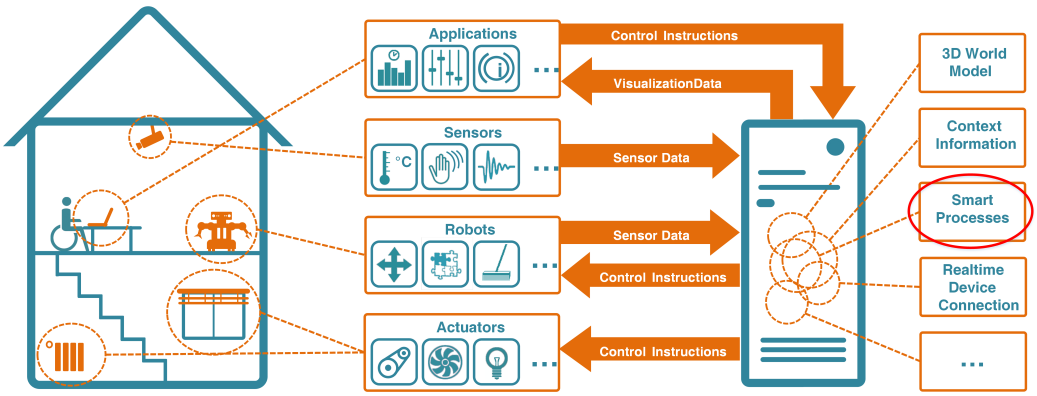


Figure 1.1.: Smart Home as Envisioned by the VICCI Research Project [Nac12].

1.2. Research Issues

The main topic of this thesis is the application of workflow technologies in CPS. We will investigate the applicability of workflows to facilitate the linking of heterogeneous CPS components across system boundaries, to increase the level of automation in CPS application scenarios, and to enable resilient and autonomous processes that are able to handle errors and other unanticipated situations themselves.

In the course of these investigations, we will discuss necessary elements of workflow languages and components of WfMSes to implement workflows for CPS. This includes questions regarding the processing of complex sensor streams, the dynamic selection of process resources, the interaction with humans, and the distributed execution of processes. CPS consist of a variety of heterogeneous resources including physical entities such as sensors, actuators, computing devices, smart objects, humans and things that interact with each other and with virtual software applications and services in a very dynamic cyber-physical environment. The focus of our inves-

tigations is to show the applicability and feasibility of using workflow technologies to describe and enact these interactions among all involved CPS entities on a more abstract business process-oriented level.

The interactions with and influence on the physical world introduce a new dimension for the modelling and execution of business processes in CPS and IoT. We will put a focus on describing and checking the effects of the process execution on the physical world and vice versa–correlating sensor data from the physical world with the processes to determine success or failure of the execution. As these interactions with the physical world introduce new error sources relevant to the process executions in CPS, we will also investigate how to detect and react to possibly undesired situations and errors. This includes the discussion of questions with respect to modelling and synchronizing the states of cyber world and physical world during process execution (*Cyber-physical Synchronization*), the automated handling of detected errors, the addition of self-management capabilities to the WfMSes for CPS, and the general extension of existing WfMSes with autonomous capabilities as part of a retrofitting process. One of the main questions of this work is related to the *Cyber-physical Synchronization* aspect for workflows depicted in Figure 1.2. In this simple example workflow from the smart home domain, the process-aware smart home control system is supposed to switch on the light in a specific room. After issuing the corresponding call to the actuator that is controlling the light and receiving a positive response from the actuator's control software, the workflow instance finishes assuming the light is switched on now. However, the light bulb maybe burnt or worn off, which the controlling actuator maybe unable to detect. This leads to an inconsistent process execution state between the cyber world (*light on*) and the physical world (*light off*), raising the questions of how to detect these kind of inconsistencies and how to remedy them automatically. From this simple scenario showing the need for considering feedback from additional data sources to verify the process executions in the physical world, we derive more complex use cases involving the interactions with humans and the state synchronization of service robots and other smart home appliances in *Cyber-physical Processes*.

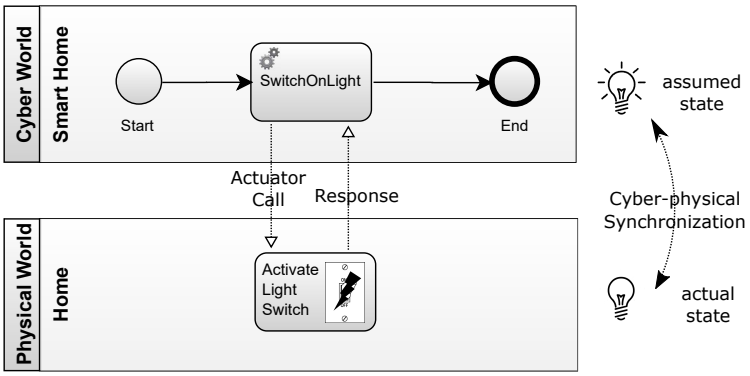


Figure 1.2.: Synchronization of the Cyber and Physical Worlds in Smart Lighting Scenario [SHS16].

When using autonomous service robots controlled by a WfMS for supportive tasks in the smart home, the robot has to be able to dynamically react to new and unforeseen situations. Simple camera-based navigation and localization done by the robot

itself is usually sufficient for navigating autonomously within known wide spaces. However, a smart home is usually full of known and unknown objects, narrow passages, furniture and other dynamically moving obstacles (cf. Figure 1.3). Fetching and driving tasks in this context require different levels of precision regarding the robot's positioning and navigation capabilities that can often only be achieved using external data to verify and adjust its position and hence, to prevent inconsistencies regarding the robot's assumed virtual and actual physical position. In addition, the robot–being a mobile device–relies on a battery and wireless network connection to receive new instructions. Due to these limitations, the robots may be temporarily unavailable for process execution or even fail the execution of process instances at runtime. In this extended example, we will investigate the workflow-based control of multiple robots and interactions with other dynamic CPS devices and data sources to add autonomic capabilities to the respective processes and WfMS and thereby, to increase the fault-tolerance and resilience of the CPS workflows.

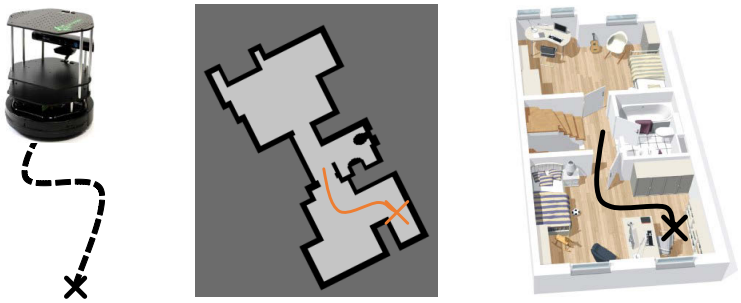


Figure 1.3.: Autonomous Robot Navigation Scenario in a Smart Home [SSAS15].

1.3. Scope & Contributions

The main domain that will be addressed to apply and evaluate the concepts developed in the context of this thesis is the *Smart Home* as a representative of a CPS. Smart homes show typical components–sensors, actuators, smart objects and humans–and behaviour that can be found in CPS. In addition to the interactions of software controlled actuators and sensors with the physical environment and objects, humans are essential entities in the smart home. Interacting with these entities requires flexible workflows that are able to adapt to new situations and to unpredictable behaviour. Many of the proposed concepts can be transferred to other smart spaces and CPS domains, though. Available consumer hardware to be used in smart home environments provides a sufficient level of open programming interfaces and non-critical behaviour to be the basis for the quantitative evaluation of our CPS workflows. When discussing aspects and concepts related to the interaction with humans, we will put no explicit focus on providing a high level of usability and with that, no user studies to evaluate the user interfaces and interactive applications presented as part of our prototypes.

When investigating the application of BPM technologies for CPS, we conduct the discussion on a more technical and architectural level regarding the workflow notations and management systems, not on the level of "classical" business pro-

cesses. We assume that a domain expert is able to identify and model the relevant workflows. Process mining, machine learning (artificial intelligence) or other ways of identifying and synthesizing parts of a workflow are out of scope of this work. Process adaptations at runtime in case of errors or other undesired behaviour are mostly done on the level of modifying the process resources as they are the main interfaces between the physical and virtual worlds and therefore close to the new error sources. Structural adaptations of processes are out of scope of this work as there has been a wide variety of related research conducted to cover this topic sufficiently.

The concepts regarding workflow languages and architectures will be described using semi-formal notations normally based on the Unified Modeling Language (UML). The application of more formal specifications and methods for the purposes of automated verification or reasoning is out of scope of this thesis. Related to that, we will only briefly discuss safety and security-related aspects (e. g., concurrent access, transactions and real-time behaviour) as they usually represent very complex orthogonal research issues that go beyond the scope of a single PhD thesis.

In short, the contributions of this PhD thesis comprise:

- a domain-independent modelling notation for executable workflows in CPS that supports the specification of the process outcome using style sheets;
- a system architecture of a distributed WfMS for CPS able to execute CPS workflows that interact with the physical world via sensors, actuators, smart objects and humans;
- a generic software component for adding feedback loops to workflows to enable cyber-physical synchronization based on the concept of *Cyber-physical Consistency* and self-management;
- a retrofitting framework for extending existing WfMSes with self-* properties.

1.4. Structure of the Thesis

This PhD thesis is structured as follows: Chapter 2 presents basic concepts from the BPM and CPS domains and deduces requirements for applying workflows in CPS that will be investigated in the course of this thesis. Chapter 3 discusses and evaluates existing BPM systems and related research with respect to the identified requirements. The main chapters containing our new concepts are aligned with the simplified BPM lifecycle (Design, Implement/Configure, Run & Adjust) [VDA13]. Chapter 4 presents our approach for modelling workflows and associated entities and resources in CPS. Chapter 5 elaborates on the basic architecture of a corresponding WfMS to execute workflows in CPS. Chapter 6 presents a generic software component and framework to equip existing WfMSes with the capabilities of cyber-physical synchronization and self-management at runtime based on feedback loops. Chapter 7 discusses the applicability and feasibility of the new concepts qualitatively and quantitatively with the help of various experiments in a smart home case study. Chapter 8 summarises the thesis and our contributions, and shows starting points for future work.

2. Workflows and Cyber-physical Systems

> "Science must begin with myths,
> and with the criticism of myths."
>
> Karl Popper

2.1. Introduction

The main theme of this thesis is the application of workflow and BPM technologies to introduce formalized processes to CPS to increase automation and have a flexible way of defining interactions among all entities in CPS. In this chapter, we introduce the most important basic concepts from the fields of workflow and BPM technologies as well as CPS and IoT. Based on the characteristics of CPS and two running examples from the smart home domain, we derive a set of new requirements that workflow languages and management systems have to fulfil in order to be used in cyber-physical environments. This list is not intended to be comprehensive with respect to all special properties of CPS, but rather a first set of requirements that emerge when investigating the new research area of *Cyber-physical Workflows* [SHS16].

2.2. Two Motivating Examples

In this thesis, we focus on the application of workflow technologies in smart homes as instances of CPS. The investigation of these environments is especially interesting as, on the one hand, smart homes feature all kinds of new "smart" devices from simple sensors to complex actuators consisting of compounds of sensors, actuators and processing units that are connected and interacting with each other (e. g., service robots [SG07]). On the other hand, the central entities within a smart home are its residents which every application and process is developed and evolved around [PWLK03]. Despite the goal of increasing the level of automation with the help of workflows, users still have to be considered as first class citizens and interaction partners. In the following, we describe two typical scenario processes in smart homes that serve as simplified running examples throughout this thesis. As they show new CPS-related behaviour and characteristics, we use these simple workflows to discuss emerging aspects and issues relevant for CPS. To describe processes on a more abstract and organisational level, we use a reduced and simplified set of Business Process Model and Notation (BPMN) 2.0 elements. In later parts of this thesis, we will apply a more technically-oriented graphical process modelling notation based on our self-developed process metamodel to have a graphical representation of a process (cf. Chapter 4) [SKNS15]. In the smart home case study conducted as part of the evaluation, we will also present more complex processes including parallel and hierarchical sequences of activities (cf. Chapter 7).

2.2.1. Morning Routine Process

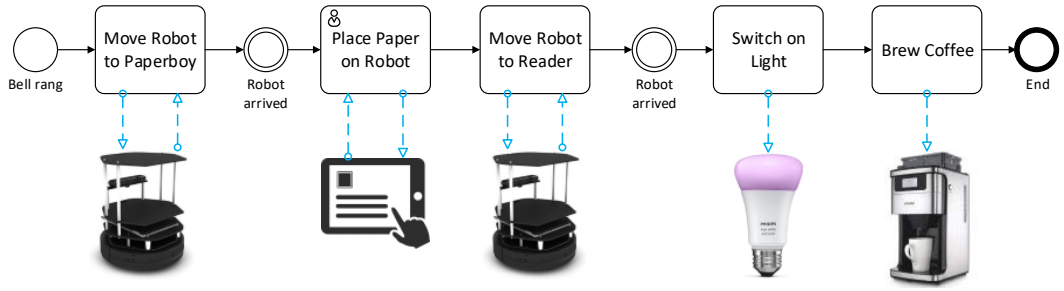


Figure 2.1.: Morning Routine Process.

The first scenario process is focused on increasing the comfort of residents in the smart home. Figure 2.1 shows this home automation process and its interactions with physical and virtual CPS entities. The process consists of an ordered sequence of activities and events to automate parts of the morning routine of a smart home resident. After the door bell rang, a service robot is instructed to drive to the paper (parcel) delivery boy. From a technical perspective, the robot continuously reports its location in the form of events to the smart home during this autonomic and asynchronous navigation task. Once the smart home control system detects that the robot has reached its destination via an explicit event, the delivery boy is notified on his smartphone to put the newspaper on the robot. After a confirmation by the delivery boy, the robot is sent to the resident to deliver the paper. Upon the successful arrival of the service robot, the light in the kitchen is switched on and the coffee maker is instructed to start brewing coffee.

2.2.2. Emergency Process

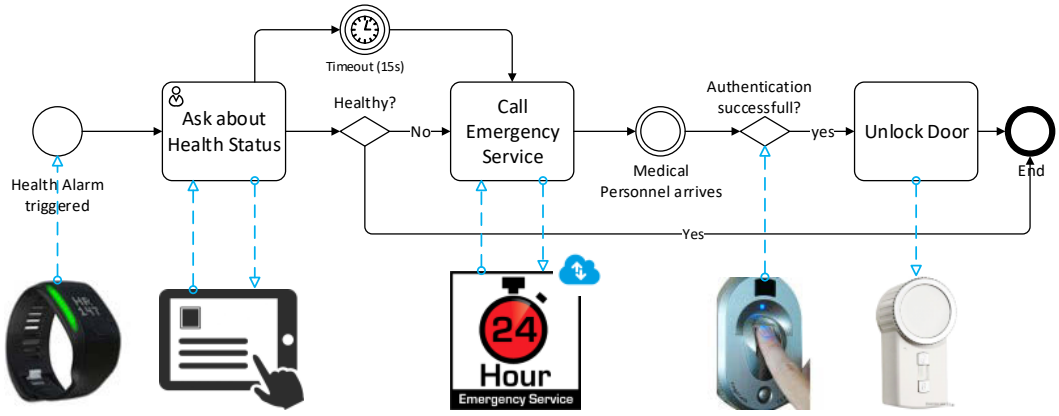


Figure 2.2.: Emergency Scenario Process.

Following the example from Dar et al. [DTRE11], the second scenario process is set in an Ambient Assisted Living (AAL) environment [SDFGB09] where elderly residents are automatically provided with assistance in emergency situations con-

cerning possible health issues. Figure 2.2 shows this process and its interactions with the CPS entities in a simplified BPMN model. A fitness tracker monitors the resident's vital signs including blood pressure and oxygen saturation. In case a sudden drop within these values appears, the resident is asked for his/her well-being on a personal interaction device. If there is a negative response or no response after a 15 seconds timeout, an emergency call is placed. Once a medic arrives and authenticates himself at the front door using a fingerprint scanner, the door is automatically opened and the medic can provide the resident in distress with medical assistance.

2.2.3. Towards Workflows in Cyber-physical Systems

The example processes described in the previous sections are two possible processes that can be designed for and implemented in a smart home to assist its residents. They show typical characteristics of CPS: the interactions between virtual software components and physical entities including humans, smart objects, actuators and sensors; and the combination of event-driven behaviour and active tasks. The processes illustrate some of the advantages the application of workflow technologies in CPS can have. Among others, workflows allow for a flexible composition of active and reactive behaviour provided by heterogeneous CPS resources integrated across individual application and system boundaries. The definition of these workflows can become significantly easier and more flexible than implementing the corresponding processes in a high-level programming language.

However, the use of workflows in CPS also comes with new challenges for the WfMSes as the interaction with the physical world holds new sources of errors and imprecisions that need to be dealt with. We will elaborate on these advantages and challenges in later sections (cf. Sections 2.5 and 2.6).

2.3. Business Process Management and Workflow Technologies

2.3.1. Basic Terminology

Figure 2.3 presents a taxonomy of important terms from the field of BPM.

Business Process A *Business Process* is defined as a"set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships." [Coa96] In this thesis, we regard CPS as the organisational structures the business processes are defined and executed in to fulfil domain-specific objectives (e. g., to provide assistance in emergency situations in AAL settings).

Process Business processes are formalized in a *Process Definition*. We use the term *Process Model* throughout this thesis as it is more related to software engineering and software modelling topics. A *Process* is the "representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process consists of a network of activities and their relationships, criteria to indicate the start and termination of the process,

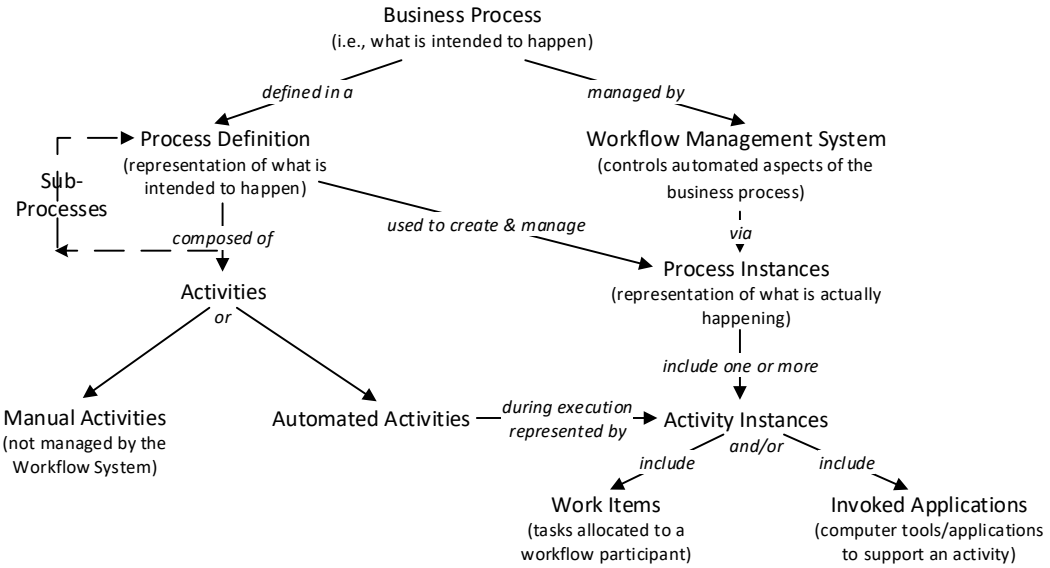


Figure 2.3.: Taxonomy of Basic BPM Terminology from [Coa96].

and information about the individual activities, such as participants, associated IT applications and data, etc." [Coa96] We refer to a *Process* as a more technical oriented, executable description of a business process in CPS. A *Process Instance* represents a particular process that is being executed or was executed by a WfMS.

Activity A process is composed of subprocesses and activities. An *Activity* is a "description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity." [Coa96] In later sections, we will also refer to an activity as *Atomic Step* to represent an automated process step that cannot be further decomposed. A manual activity performed by a person will also be called *Human Task* [AAD+07].

Workflow A *Workflow* can be regarded as the "automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [Coa96] When talking about workflows, we refer to a more abstract high-level description of sequences of activities, events and tasks of a process not focussing on implementation details.

Workflow Management System A *WfMS* is a "system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process, interact with workflow participants and, where required, invoke the use of IT tools and applications." [Coa96] We will also use the term Business Process Management System (BPMS) as a synonym for a WfMS.

Work Item A *Work Item* is the "representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance." [Coa96] A *Worklist* contains the work items of a particular workflow participant or group of participants. The *Worklist Handler* is responsible for managing the interaction between users and the worklist maintained by the WfMS. In the course of this thesis, we will refer to a process *Resource* as a more general term for describing the worklist participant as it not only refers to users but also to devices and software applications involved in the execution of a process activity in CPS [DLRM+13].

Event In the context of BPM, an *Event* represents an "occurrence of a particular condition (which may be internal or external to the workflow management system) which causes the workflow management software to take one or more actions." [Coa96] We will also consider the arrival of messages, new data from sensors and actuators of CPS, and the detection of new situations as events–with or without relevance for the process execution [Tal08, OHG17].

2.3.2. The BPM Lifecycle

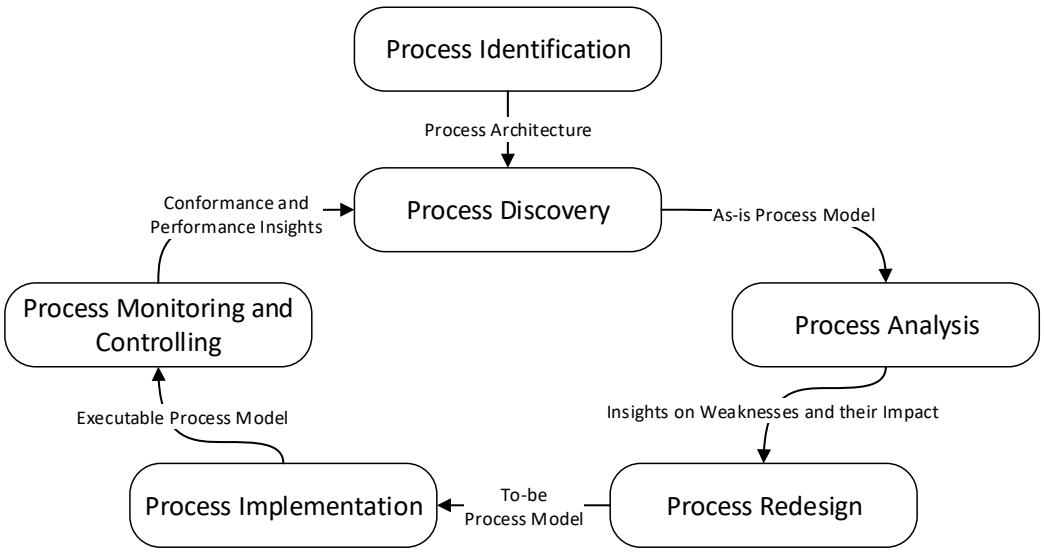


Figure 2.4.: The BPM Lifecycle from [DLRM+13].

One of the key principles when introducing the concept of business processes and designing a Process-aware Information System (PAIS) [DVdATH05] is the *BPM Lifecycle* [DLRM+13], which follows the Deming Plan–Do–Check–Act (PDCA) Cycle [Joh02]. It is depicted in Figure 2.4. First, relevant processes within an organization or–in the context of this thesis–within the CPS have to be identified on an abstract level. From these processes and their interrelations, a general *Process Architecture* describing the processes, their dependencies and interactions with each other is derived. The individual processes are then formalized in a process model as they are currently carried out (*Process Discovery*). Following an analysis of these processes evaluating weaknesses and their impacts, the processes are re-

designed resulting in *to-be* process models. The next phase deals with the technical implementation of the respective processes that leads to the derivation of executable process models. During the execution of instances of executable process models, the *Process Monitoring and Controlling* phase leads to insights about the conformance and performance of the respective processes. The BPM lifecycle then repeats itself with the evolutionary discovery of new and existing processes based on the results of the process monitoring and controlling. In the context of this thesis, we assume that experts in the individual CPS domain are responsible for the initial identification, discovery and implementation of the relevant processes. The goal is to automate parts of the implementation, monitoring and controlling, as well as analysis and redesign at runtime in order to deal with unanticipated errors and situations.

2.3.3. Workflow Languages

Workflow languages are used to describe and define workflows in a formal way. They range from textual notations that are more implementation oriented (e. g., Web Services Business Process Execution Language (WS-BPEL) [OAS14]), to more formal workflow languages (e. g., Yet Another Workflow Language (YAWL) [vdAtH05]) and graphical modelling notations with execution semantics (e. g., BPMN 2.0 [Mod11]) or purely symbolic notations (e. g., PICTURE for processes in public administrations [BPR07]). These languages usually include a common set of elements to describe the control flow as well as the data flow within a process–among activities and subprocesses along transitions between one or more activities and logic elements. Messages, events and resources also belong to the important elements of a workflow notation [SKNS13]. With these elements, simple Event–Condition–Action (ECA) rules and Event-driven Process Chains (EPCs) [STA05] as well as very complex operational and cross-organizational processes can be defined imperatively. Another form of modelling workflows is in a declarative way [vDAPS09], which relies on a more loose specification of workflow activities, not defining a specific sequence of executions but rather constraints limiting the order the activities can be executed (e. g., within the DECLARE language [RSS13]).

BPMN 2.0

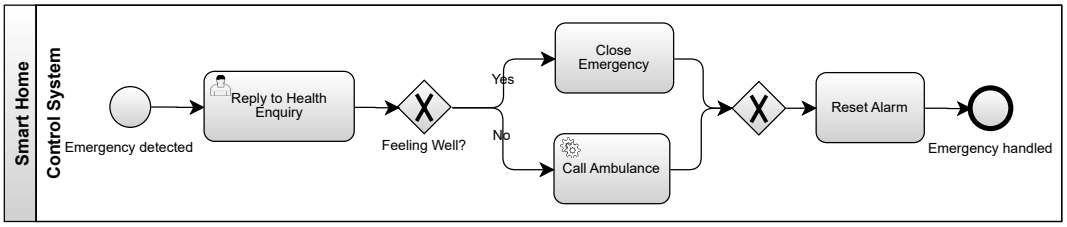


Figure 2.5.: Partial Emergency Scenario Process in BPMN 2.0.

BPMN evolved from a purely graphical symbolic business process modelling language to the de facto standard language for describing organizational business processes in its current version 2.0, which also features execution semantics [Mod11]. It supports a rich set of various types of activities, gateways, events, messages, data

objects, pools and lanes. Figure 2.5 shows a simplified excerpt from the emergency scenario process described in Section 2.2.2 in BPMN 2.0. The smart home is depicted as a *Pool*, which is used to represent whole organizations. A pool contains *Lanes* (here: the control system), which describe who is executing a specific set of tasks. When an emergency situation is detected as an event, the user has to reply to the health enquiry. In case of an emergency, the ambulance is called–otherwise a false alarm is assumed and the emergency situation is closed, and the alarm is then reset. A critical evaluation of BPMN with respect to the workflow patterns [vDATHKB03] and its suitability to model business processes can be found in [WvdAD+06, Bör12]. Some of the criticisms of BPMN include the large and complex set of modelling elements that lead to an increased difficulty of creating unambiguous process models and the lack of standard workflow engine implementations for BPMN 2.0.

WS-BPEL

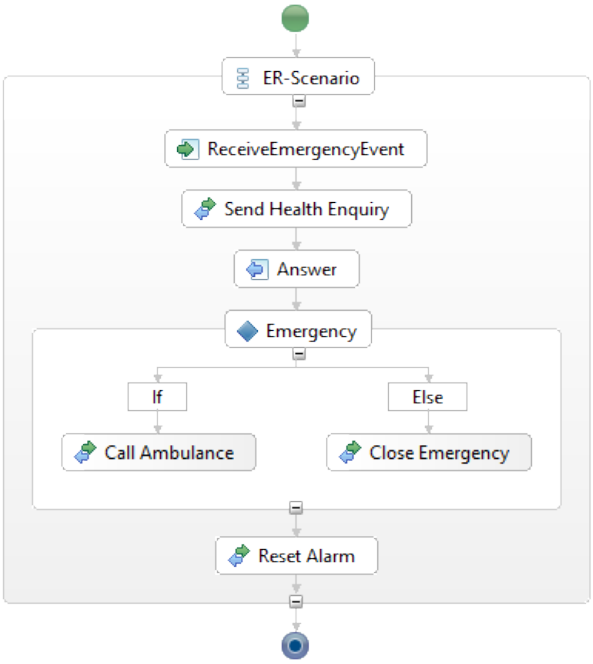


Figure 2.6.: Partial Emergency Scenario Process in WS-BPEL.

Before BPMN, WS-BPEL [OAS14] was widely used as an Extensible Markup Language (XML)-based language to describe and enact business process in distributed service-oriented architectures. WS-BPEL has a stronger focus on describing executable processes in IT systems, which leads to a certain gap between this technical specification and the description of the actual underlying business process [PDB+08]. Over the years, a vast variety of extensions were introduced to WS-BPEL to reflect various additional aspects of business processes (e. g., the *BPEL4People* extension to model human activities [KKL+05]). Figure 2.6 shows an excerpt from the emergency scenario process in a graphical notation for WS-BPEL. As its focus is on web service

orchestration, the underlying textual process specification contains many parameters and attributes regarding the actual service calls and data to be exchanged.

YAWL

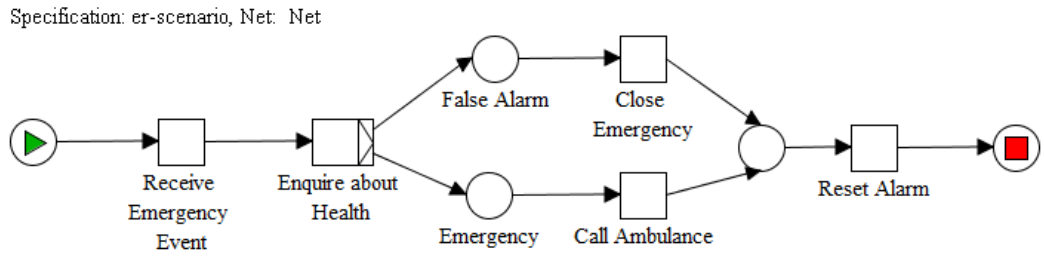


Figure 2.7.: Partial Emergency Scenario Process in YAWL.

Compared to BPMN and WS-BPEL, which are widely used in organizational contexts, YAWL [vdAtH05] is mostly applied in academic settings. YAWL was developed with a focus on implementing the workflow patterns suggested by Van Der Aalst [vDATHKB03]. Processes created with YAWL are formal, executable processes as YAWL relies on workflow nets, which are based on Petri nets [VdA98]. Central modelling elements are places and transitions with preconditions and postconditions. During execution of a YAWL process, tokens move through the workflow net from places to places after triggering transitions. Due to being founded on Petri nets, these processes can be analysed more formally with respect to the properties of Petri nets [AAH98]. Figure 2.7 presents an excerpt from the emergency scenario process as a YAWL process.

2.3.4. Workflow Management System

The WfMS is responsible for enacting instances of executable process models. Figure 2.8 shows the essential components of a BPMS as proposed in [DLRM+13] and [GdV98]. The *Process Modelling Tool* is used to create an abstract formal description of a process based on a workflow language. The resulting process models are either instantiated and executed by the *Execution Engine*, which is the core component of the BPMS or they are stored in a *Process Model Repository* for later use. The *Worklist Handler* manages tasks assigned to the process resources during execution of a process instance. Interaction with the execution engine is enabled by various *Monitoring & Administration Tools*, which allow for the control and visualization of the process execution. During the execution of processes, the execution engine produces *Execution Logs* that can be inspected with the help of the monitoring tools or used for process discovery and redesign based on *Process Mining* approaches [VDAADM+11]. The execution engines mostly interacts with external services to enact the business processes. There exists a large variety of industrial and academic WfMSes based on various workflow languages. We present a brief overview and evaluation of some systems in Section 3.2. The majority of current BPMS enact workflows and communicate with external applications and services based on the SOA-paradigm.

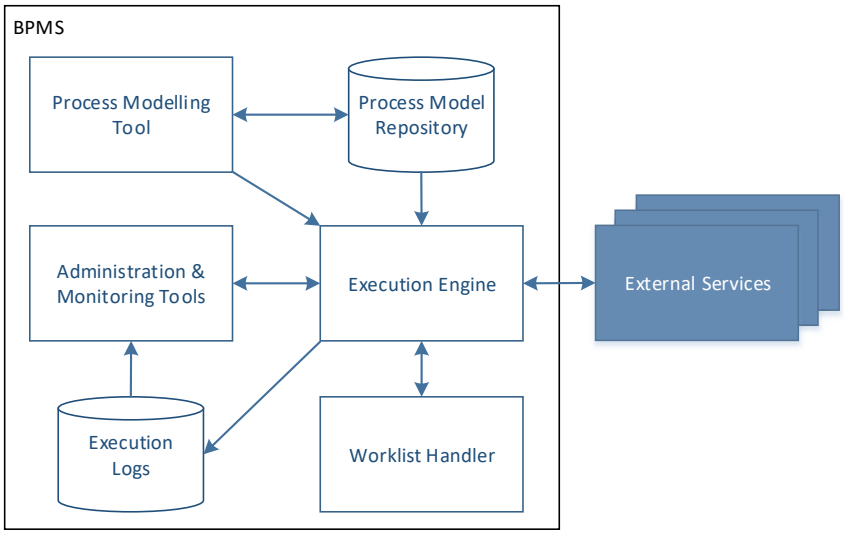


Figure 2.8.: The Architecture of a BPMS from [DLRM+13].

2.3.5. Service-oriented Architectures

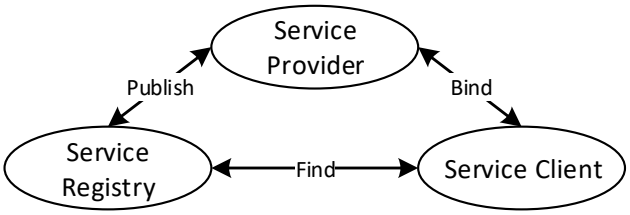


Figure 2.9.: The Basic SOA from [Pap03].

SOAs are the main principles that BPMSes rely on as one of their main purposes is the orchestration and choreography of SOA-based service invocations within and across service-based enterprise applications. When executing a process, the BPMS invokes functionality from various internal or external software components that are represented and implemented as services in SOAs hosted by a web server. In principle, SOAs are "a way of reorganizing a portfolio of previously siloed software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols." [Pap03] This architecture is especially suitable when multiple applications using heterogeneous technologies and platforms need to communicate with each other in a flexible and decoupled way.

Figure 2.9 shows a basic high level view on SOAs. A *Service* usually represents a business function implemented in software. It is accompanied by a formal interface description (e. g, using the Web Services Description Language (WSDL) or Web Application Description Language (WADL)) that describes how to invoke the service–technically the set of operations provided by the service. These service descriptions are published by the respective *Service Provider* to a *Service Registry*. *Service Clients* can query the registries to find suitable services and bind the respec-

tive service endpoints to their applications. The formal description of a service's interfaces decouples the service's offered functionality from its actual implementation, which leads to more flexible services and service compositions as clients can invoke the services based in the provided interfaces in a technology agnostic way [Pap03]. These advantages made SOA the predominant paradigm for distributed business applications and WfMSes as well as the IoT [GIM11]. Here the Representational State Transfer (REST) architecture has become the prevalent approach for implementing web services. More elaborations on individual SOA-related aspects and its relation to the BPM domain can be found in [WCL+05, PTDL07, Erl05].

SOA-based Deployment Models for IoT

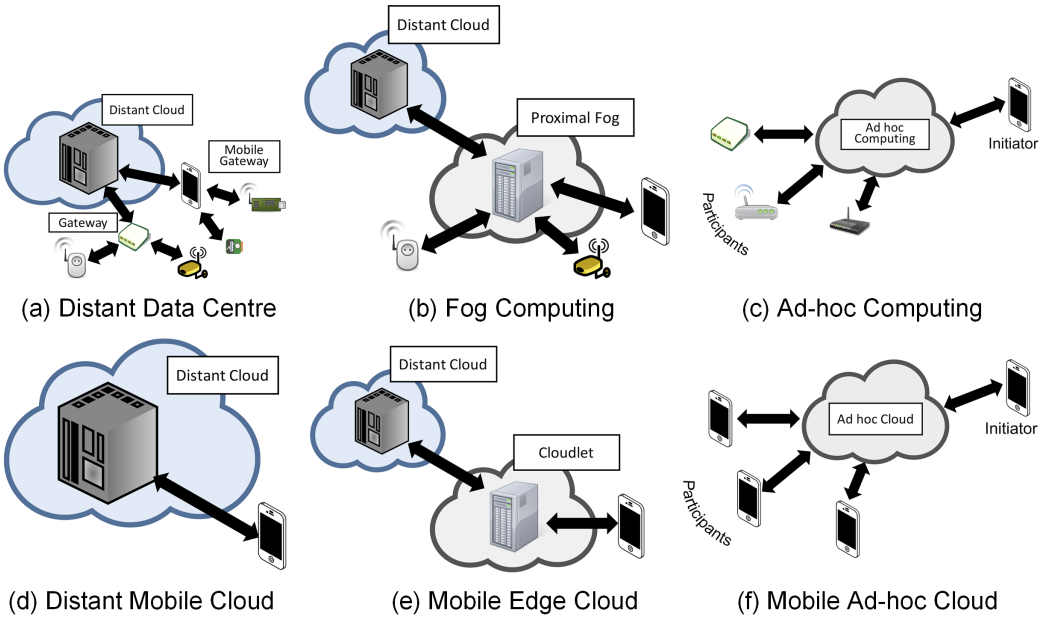


Figure 2.10.: Deployment Models for IoT from [CSB16].

Nowadays, various new computing paradigms and deployment models for mostly service-based architectures exist that extend the classical client-server model to more advanced system architectures based on virtualization of resources and locality of processing and information. Figure 2.10 gives an overview of current deployment models discussed by Chang et al. and Niroshinie et al. in the context of BPMSes, IoT and mobile Cloud computing [CSB16, NSW13].

- The *Distant Data Centre* model (cf. Figure 2.10(a)) describes the classical Cloud computing approach with mobile and stationary devices communicating via network gateways with remote and virtualized servers and resources on different service levels (*XaaS* [DFZ+15]).

- The *Fog Computing* model (cf. Figure 2.10(b)) describes the use of computing resources of the network infrastructure between distant Cloud servers and

mobile or stationary clients (edge devices) often in a hierarchical network structure. Network gateways enable the proximal preprocessing of data from edge devices in a Fog before it is being sent to the Cloud [BMZA12].

- The *Ad-hoc Computing* model (cf. Figure 2.10(c)) describes the use of local computing resources to create a local Cloud often initiated by a mobile or stationary edge device for local ad-hoc processing of data and task execution (cf. *Mist Computing* [Cor16] and *Dew Computing* [SDA+15]).
- The *Distant Mobile Cloud* (cf. Figure 2.10(d)) refers to the Distant Data Centre model of mobile devices using resources and services of distant Cloud servers.
- The *Mobile Edge Cloud* (cf. Figure 2.10(e)) refers to the Fog Computing model of network gateways providing micro-services in a small Cloud (*Cloudlet*) for proximal data preprocessing with mobile devices and transferring the results to the distant Cloud [PNC+14].
- The *Mobile Ad-hoc Cloud* (cf. Figure 2.10(f)) refers to the Ad-hoc Computing model of mobile edge devices forming a local ad-hoc Cloud for local data processing.

2.4. Cyber-physical Systems

2.4.1. Basic Terminology

Cyber-physical Systems According to Lee, *Cyber-physical Systems* (CPS) are "integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa." [Lee08] CPS combine software, sensors and physics via actuators that act independently, cooperative or in the form of System of Systems (SoS) composed of interconnected autonomous systems [SS12].

Internet of Things The *Internet of Things* (IoT) can be regarded as the "worldwide network of interconnected objects uniquely addressable based on standard communication protocols." [GBMP13] In this context, *Things* "are active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information". [GBMP13]

CPS vs. IoT The terms CPS and IoT are often used as synonyms. In this thesis, we follow the idea of Chen that the physical entities of the IoT are represented by their corresponding virtual entities (*Digital Twins*) that are interconnected via the Internet (e. g., through web services) [Che10]. As shown in Figure 2.11, the unions of the physical entities $(P_1, ..., P_n)$ and their corresponding virtual representations in the cyber world $(C_1, ..., C_n)$ are regarded as CPS. The focus of IoT is on the perspective of *Things* and *Devices* and enabling their interconnectivity through wired or wireless technologies (*network-centric*). CPS on the other hand, put their focus on

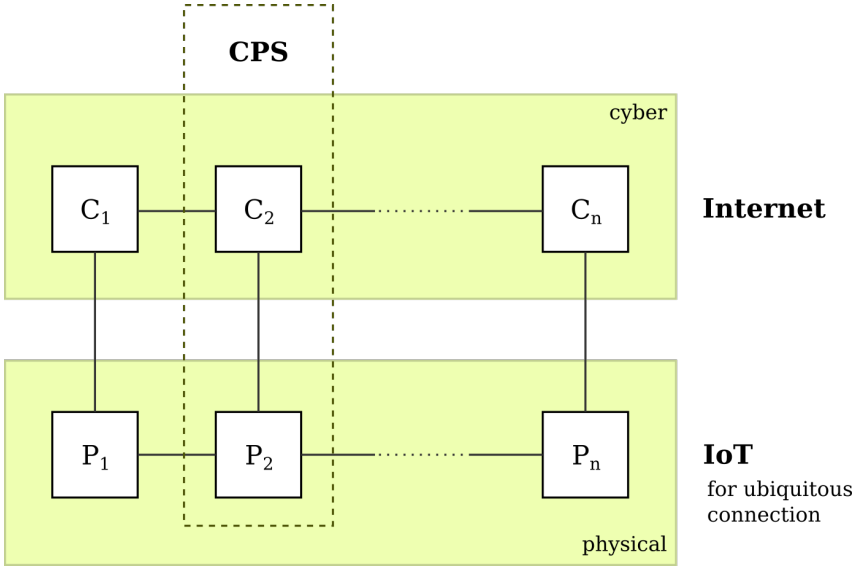


Figure 2.11.: Relation between IoT and CPS from [Che10].

the *System* perspective to enable interactions with the physical world via respective algorithms (*component-centric*). The IoT technologies developed for interconnecting things and devices leverage the development of CPS.

Smart Home The focus of this thesis is on application scenarios within smart homes as instances of CPS. A *Smart Home* can be defined as a "residence equipped with computing and information technology which anticipates and responds to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond." [Ald03] A special form of a smart home is an AAL environment, which aims at supporting people with disabilities and health issues as well as elderly people in living a self-determined and independent life [SDFGB09].

2.4.2. Context in CPS

An important characteristic of CPS is that the systems and their components are showing a highly context-dependant behaviour, i. e., available functionality and interactions of CPS components with other entities are influenced by intrinsic and extrinsic context factors [Bro13]. A common definition of *Context* from the field of ubiquitous computing, which can also be applied within the scope of this thesis is given by Dey: "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [Dey01] We extend the notion of an *entity* to also comprise CPS devices, things and workflows.

A proposition of an exemplary context taxonomy can be found in [KKS11]. This work classifies a user's context into the *Interaction Context*, *Temporal Context*, *Spatial Context*, *Task Context*, *Physical Context* and *Socio-technical Context*. For our

smart home use cases, we mostly refer to context factors relating to the spatial, temporal and physical context as they are highly relevant for CPS. We base the context model used within later concepts on an ontology to describe relevant context factors of CPS entities and relations among them [HSKS16b]. In this work, context-awareness is considered to be a property of CPS. However, we put no explicit focus on realising context-adaptive behaviour as part of the WfMS's features. More detailed discussions of context related to business processes and workflow systems can be found in [TGD+08, SLI08, AFG+07, SWC+18].

2.4.3. A High-level CPS Architecture

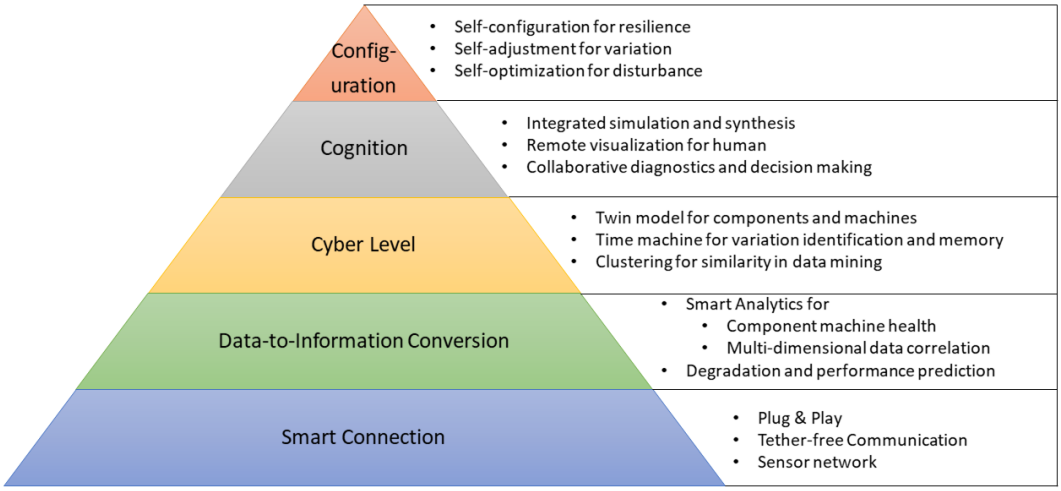


Figure 2.12.: Architecture for the Implementation of CPS from [LBK15].

The development of CPS is a highly complex topic, which comprises a wide variety of research questions from different areas. From various propositions of architectures and approaches for developing and implementing CPS, we pick the high-level architectural view on CPS for Industry 4.0 proposed by Lee et al. [LBK15] as it provides a suitable level of abstraction for discussing challenges and concepts in the context of this thesis. Figure 2.12 shows the five level architecture.

On the bottom *Smart Connection* level, devices are seamlessly connected into a network of computers, sensors, actuators, machines, robots and things. Plug & Play functionality allows for the easy addition and removal of known and unknown devices. The *Data-to-information Conversion* level provides means for smart analytics of device and component health, predictive maintenance, human behaviour and other multi-dimensional data correlations. The *Cyber* level comprises twin models for components and machines as comprehensive virtual representations of the physical entities of the CPS (*Digital Twins* [BR16]). It also includes a time machine mechanism to explore histories and variations of components and processes as well as means of clustering similar data for data mining. On the *Cognition* level, integrated simulation and synthesis of components, knowledge, situations and processes have to be provided. Cognition also refers to enabling remote visualization for users to understand the CPS as well as the collaborative diagnostics and decision making of software agents within the CPS. Based on these levels and concepts, the top level

comprises *Configuration* of the CPS based on self-* mechanisms (cf. Section 2.4.5) to create variations, to react to errors and disturbances, and to enable resilience in general, which is one of the main goals of this thesis.

2.4.4. IoT Reference Model

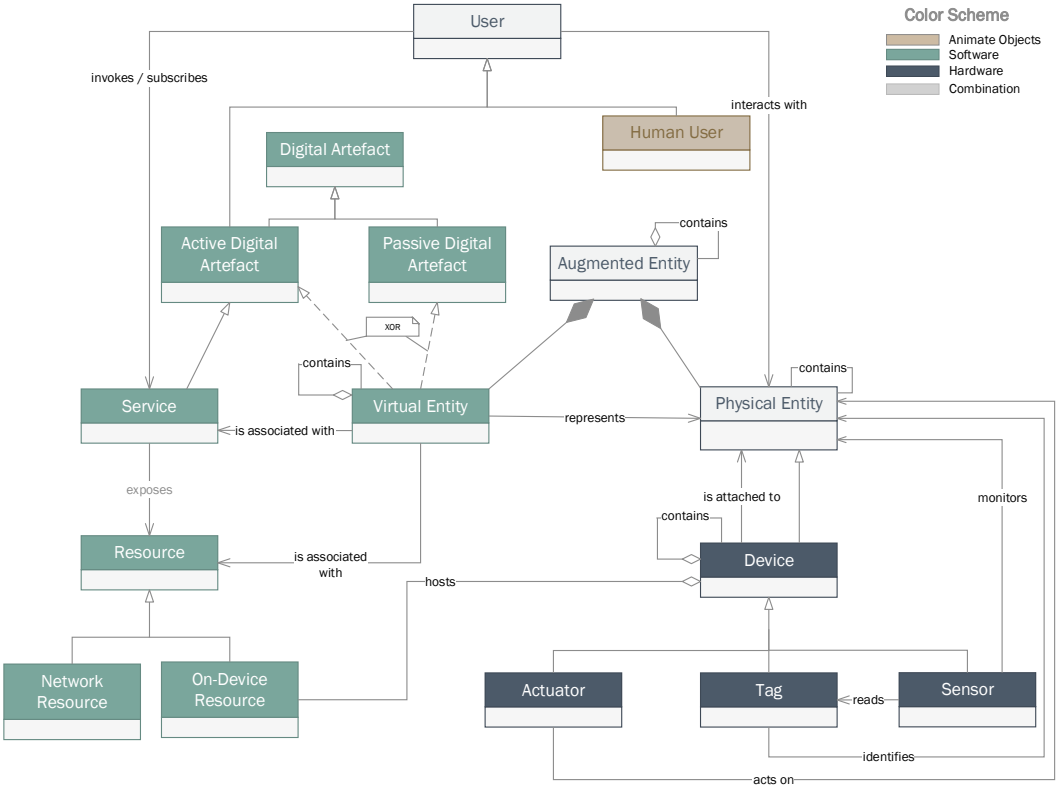


Figure 2.13.: IoT Reference Model from [BBDL+13].

The IoT Reference Model by Bauer et al. [BBDL+13] provides a suitable basis for our work to describe the entities in CPS and IoT environments and their relations. The core of this model is depicted in Figure 2.13. It classifies the entities into physical *Hardware* components (Sensors, Actuators, Tags); virtual *Software Components* (Artefacts) that are exposed and invoked via services; *Combinations* of hardware and software; and *Animate Objects* (Humans, Animals). Physical and virtual entities can be composed of multiple components. Actuators influence physical entities, sensors monitor these entities and are also able to read tags to interact with physical objects. With this model, the authors propose to use *IoT Services* [TMS+12] as basic mechanism to interact with virtual and physical entities, either via active invocations or passive subscriptions. Services are associated with virtual entities, which represent the corresponding physical entities. The attached IoT devices may host the respective services themselves (On-demand Resources) or their functionality is offered via external services (Network Resource). We will follow this central *IoT service*-based approach in our own concepts and architecture for implementing workflows and the corresponding infrastructure of CPS [MRM13].

2.4.5. Self-aware and Self-managed Computing Systems

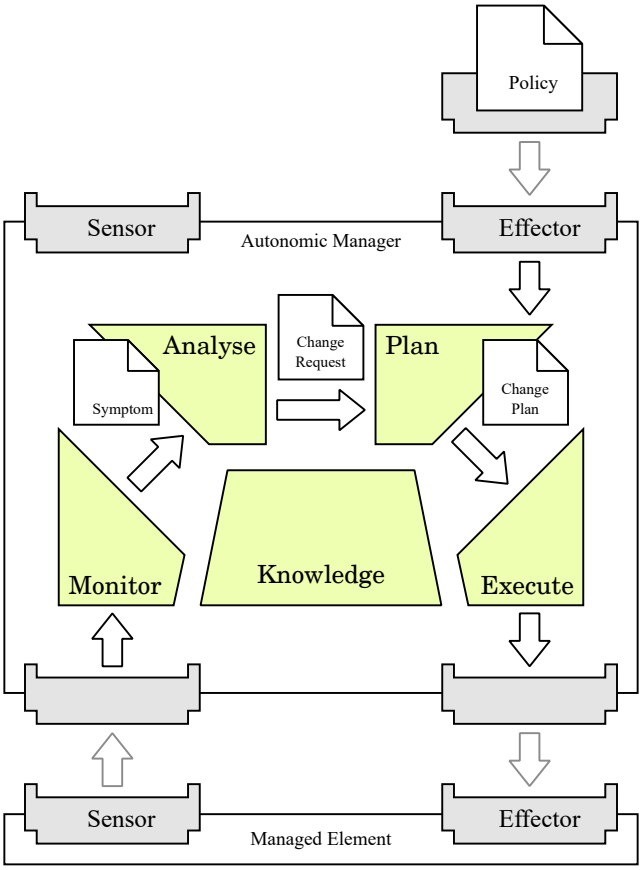


Figure 2.14.: Functional Details of the MAPE-K-based Autonomic Manager from [IBM05].

As shown in the top level of Figure 2.11, self-configuration, self-adjustment and self-optimization are important capabilities of CPS, which can be subsumed under the terms *Self-aware Computing* [Bro13] and *Self-managed Systems* [KM07]. According to Kounev et al., *Self-aware Computing Systems* "are systems that: 1) learn models capturing knowledge about themselves and their environment (such as their structure, design, state, possible actions, and run-time behavior) on an ongoing basis and 2) reason using the models (for example predict, analyze, consider, plan) enabling them to act based on their knowledge and reasoning (for example explore, explain, report, suggest, self-adapt, or impact their environment) in accordance with higher-level goals, which may also be subject to change." [KLB+17] During the course of this thesis, we will mostly refer to the term *Self-management* for WfMSes as a generalization of various concepts related to self-aware computing and self-* (self-x) properties for CPS workflows (e. g., self-adaptation, self-healing and self-optimization) [KM07].

One of the most applied principles to enable self-* capabilities and autonomic behaviour of software components is the MAPE-K feedback loop [MSW16]. The functional details of a MAPE-K-based autonomic manager are depicted in Figure 2.14 [IBM05]. The *Autonomic Manager* is used to analyse and adapt a specific

software component (*Managed Element*). This managed element provides sensor information about its current states and operations that are collected within the *Monitor* phase of the autonomic manager. Relevant changes within this data–called *Symptom*–are evaluated according to specific rules and policies in the *Analyse* phase. In case an undesired behaviour can be detected based on the symptoms, a change request is created and transferred to the *Plan* phase to find a suitable *Change Plan*, which contains compensation actions to adapt the managed element. This change plan is then enacted in the *Execute* phase of the autonomic manager by invoking the respective operations on the *Effectors* of the managed elements. The central component of the autonomic manager is the *Knowledge Base*, which contains all relevant data, rules, models and context information. As proposed by Kramer and Magee, the individual phases of the MAPE-K loop can also be controlled by an autonomic manager to increase the adaptivity of the managed system [KM07].

2.4.6. Properties of CPS

From the previous elaborations and related research, we derive the following list of properties of CPS that are relevant for this thesis, but not intended to be comprehensive. This list should be extended and detailed in future work with respect to the respective application domains and the scope of the future research. Broader discussions of properties and challenges of CPS development can be found in [Bro13, Lee08, GPGV14].

P1: Cyber-physical Interactions The main characteristic and novel feature of CPS is the mutual interaction between the virtual world and the physical world. Software controlled actuators influence physical entities, which again can effect virtual processes and software applications. Sensors are able to measure physical properties that can be evaluated by software applications controlling the CPS. Virtual representations of physical entities (*Digital Twins*/*Cyber-physical Objects* [PLM16]) have to exist to enable this form of cyber-physical feedback loop. [Lee08]

P2: Hierarchical Device Structures The basic building blocks of CPS are sensors and actuators, which are the interfaces between the physical and virtual worlds; as well as computing units to process data in between. The complexity of devices in CPS ranges from simple sensors to complex combinations of various sensors, actuators and computing units possibly interacting with Cloud servers during operation (e. g, service robots as described in the first smart home scenario process in Section 2.2.1). In CPS, these devices are usually organized hierarchically (i. e., a device consists of multiple subdevices, which again may be composed of other devices) [SS12]. Current research also discusses the next hierarchy levels in the form of Cyber-physical Systems of Systems (CPSoS) [CBF+16, CSB16].

P3: Limited Physical Resources In comparison with computations and interactions in purely virtual environments (e. g., on Cloud servers or computing grids), where usually a large pool of computational resources is available and scalability is rather easy to achieve (e. g, through virtualization), resources in the physical world are scarce and much more constraint. This will often lead to concurrent physical

processes that intend to interact with the same physical resources and CPS devices. [Lee08]

P4: Dynamic Availability of Devices As CPS consist of a variety of heterogeneous devices also comprising embedded computers and mobile devices, there are limitations regarding the availability of computational resources, too. Embedded and mobile computers are usually constraint on memory, computing power, power supply and network connectivity, which results in very dynamic and loosely coupled network structures with unreliable devices and fluctuating availability of resources and services. [CDB+12, DTB+15, SGCG18]

P5: Physical Error Sources From the interactions with the physical world, new source for errors emerge that need to be considered in CPS control software. As mentioned before, physical world resources as well as computational resources are constraint. Physical processes and interactions are usually more asynchronous, more imprecise and take longer to execute than computations in the virtual world. In addition, physical resources and objects may show signs of degradation, they may break and they may be influenced by other physical entities. [Lee08, SGLW08]

P6: Context-dependant Behaviour Context plays an important role in CPS. The behaviour of CPS entities depends on various physical and virtual context factors as well as on interactions with other entities in these very dynamic environments. [Bro13, WZZ+14]

P7: Human Interactions In the scope of this thesis, we regard application scenarios for CPS that are focussed on supporting and interacting with humans. They are important first class entities in CPS that the physical and virtual processes interact with. [BCG12]

P8: Safety-critical Behaviour The real world interactions with humans and physical objects in CPS may be safety-critical to some extend, requiring appropriate means of ensuring safety and real-time behaviour. Despite being out of focus of this thesis, we will discuss real-time and safety in later chapters as these are inherent properties of CPS. [CBF+16]

P9: Unanticipated Behaviour As CPS consist of various heterogeneous devices and other systems whose interactions with each other and with the physical environment have not been fully anticipated at design time, there is emergent behaviour to be observed from the interactions of the constituent CPS systems/components and other entitites (e. g., humans and things) at runtime, especially when discussing CPSoS. [CBF+16, BCG12]

P10: Autonomy Due to CPS being composed of various subsystems that are usually organized in hierarchical structures, there are no centralized control entities any more as opposed to classical enterprise applications. The distributed nature of CPS shows a higher degree of decentralization and autonomy–and with that self-awareness–of individual subsystems/components in CPS. [CDB+12, Bro13]

2.5. Workflows in CPS

The main topic of this thesis is the application of workflow technologies to implement and control high-level processes in CPS. In the following, we will call these processes *CPS Workflows* and workflow-enabled CPS *Process-aware Cyber-physical Systems (PACPS)*. From the findings in previous sections, we see that BPM technologies have matured in the field of service-oriented computing to automate business processes in enterprise contexts. However, existing workflow languages and management systems have not been designed to be used in CPS and therefore do not consider the specific properties of CPS, which is why their suitability needs to be evaluated and probably new concepts have to be introduced to design PACPS (cf. Chapter 3).

2.5.1. Process Levels in CPS

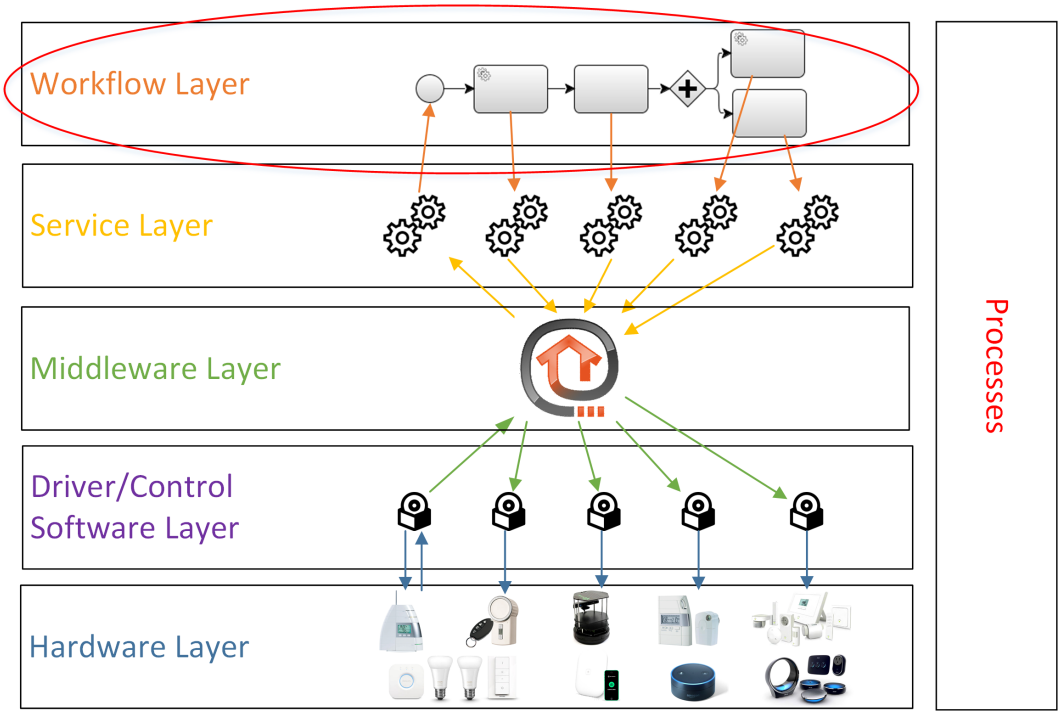


Figure 2.15.: Workflows on Top of Common Implementation Layers of CPS [BSMD11].

Figure 2.15 presents a typical layered architecture for implementing CPS on the technical level that will serve as basis for the following investigations [BSMD11]. The *Hardware Layer* comprises all CPS devices (sensors, actuators, computers, etc.) that are able to actively interact with the physical environment. This is usually a highly heterogeneous set of IoT devices from various vendors. The *Driver/Control Software Layer* adds proprietary software to control the specific hardware components to the stack. This layer also includes purely virtual (cyber) software components and applications. To make the CPS devices' and software components' functionalities remotely accessible, they can be either connected via device/component specific adapters to a middleware (e. g., the Open Home Automation Bus (openHAB) IoT

middleware for home automation [SZZ14]) on the *Middleware Layer*, which provides a homogeneous programming interface for accessing the devices and components; or they can be directly augmented by web services (e. g, based on the REST or Simple Object Access Protocol (SOAP) technologies) on the *Service Layer*. After unification of the heterogeneous device control software and other software components using proprietary or standardized protocols, the middleware also enables this service-based access to the CPS devices and other distributed software components on the *Service Layer*. The main task of the *Workflow Layer* in PACPS is then the high-level orchestration of service calls to the respective web services based on workflow/process definitions.

As shown in Figure 2.15, *Processes* can be found on each layer–from physical and electrical processes in the hardware, to operating system and application level processes on the device/control software layer, processes on the respective web servers and middleware, up to high-level executable processes among the distributed systems on the workflow layer [Lee08, AIM10]. On the upper layers, the processes are usually synchronous or asynchronous and based on discrete events and actions. On the lower layers, continuous and synchronous control processes are usually the predominant form of processes [DLV12], which are also used to implement more safety-critical functionality of a CPS device. The properties of the processes on the individual layers as well as their cross-layer influences and interfaces for cross-layer interactions have to be investigated in more detail in future work to identify dependencies among processes that will influence the design and future developments of the workflow layer.

The service robot used in the *Morning Routine* scenario (cf. Section 2.2.1) is a good example for these multi-layer processes: during autonomous navigation based on Simultaneous Localization and Mapping (SLAM), the robot uses continuous processes close to the hardware to control its locomotion engine and to evaluate the camera stream to derive navigation instructions; the more abstract functionality to drive to a certain position in a room is exposed via a high-level asynchronous service call based on a Robot Operating System (ROS) service, which emits events in case the robot arrived or got stuck. In the scenario, this high-level service is invoked by the WfMS as part of the execution of the *Morning Routine* process. The focus of our investigations is on the *Workflow Layer* (cf. Figure 2.15) and the implementation of PACPS. However, aspects considering the other layers will also be discussed.

2.5.2. Advantages of Using Workflows in CPS

Besides achieving a general level of repeatability, the introduction of a dedicated workflow layer and with that, the application of workflows to orchestrate high-level processes among the devices, things and humans in PACPS as proposed in the previous sections offers the following advantages. The list shows advantages of workflows for PACPS in the context of this thesis, mostly over the use of "traditional" higher-order programming languages and approaches in CPS. It is not meant to cover all possible advantages, though.

- **Increased Automation:** The inherent purpose of workflows is to automate repetitive tasks in different domains. By applying workflows to CPS, the level of automation in cyber-physical environments can be increased. Advantages

such as a more efficient resource usage, higher product quality, increased comfort, higher safety, and cost savings can be often observed associated with this increased level of automation.

- **High-level Programming:** Workflows can be viewed as simple high-level programs defining the flow of activations, messages, events and data among services in CPS. They are usually less complex than higher-order programming languages.

- **Flexible Combination of Functionality:** The high-level programming of workflows allows for a flexible combination of available functionality in the sense of EAI. Workflows can be programmed and changed faster than hard-wired programs created with higher-order programming languages.

- **End-user Programming:** Workflow languages are usually designed to be used by domain experts and not necessarily by software developers. Depending on the complexity of the workflow language and the respective modelling tools, also end-users can be enabled to create simple executable workflows using graphical editors and drag & drop interactions.

- **Cross-application/Device/System Orchestration:** As workflows are usually implemented on top of web services, a flexible integration, orchestration and combination of functionality across the borders of single systems, devices and applications is possible.

- **Easy Integration of Heterogeneous Systems:** With web services being the unifying layer to access the heterogeneous subsystems/devices of CPS, workflows can be used to achieve an easy integration and coupling of existing systems independent of their underlying technologies.

- **No Alterations to "Original" Systems:** The existing functionality of the individual CPS components is exposed via web services. These web service can be used to build high-level programs in the form of workflows. This approach allows for creating new functionality, applications and processes without the necessity of modifying the underlying components' control software or services.

- **Complex Active and Reactive Behaviour:** As workflows support the concepts of asynchronous messages and events as well as active service invocations, active and reactive behaviour can be combined to realize more complex system functionalities, program logic and behaviour within a workflow.

- **Documentation (Traceability):** Workflows are usually specified based on a formal metamodel (workflow language). The corresponding workflow models combined with instance data from process monitors (histories, logs) allow for the documentation of relevant processes (*Process Architecture*) as well as tracing the progress of process instances.

- **Analysis/Optimization:** The formal specification of workflows also facilitates the analysis and optimization of individual processes, instances as well as of the overall process architecture (*Process Landscape*).

2.5.3. Challenges of Using Workflows in CPS

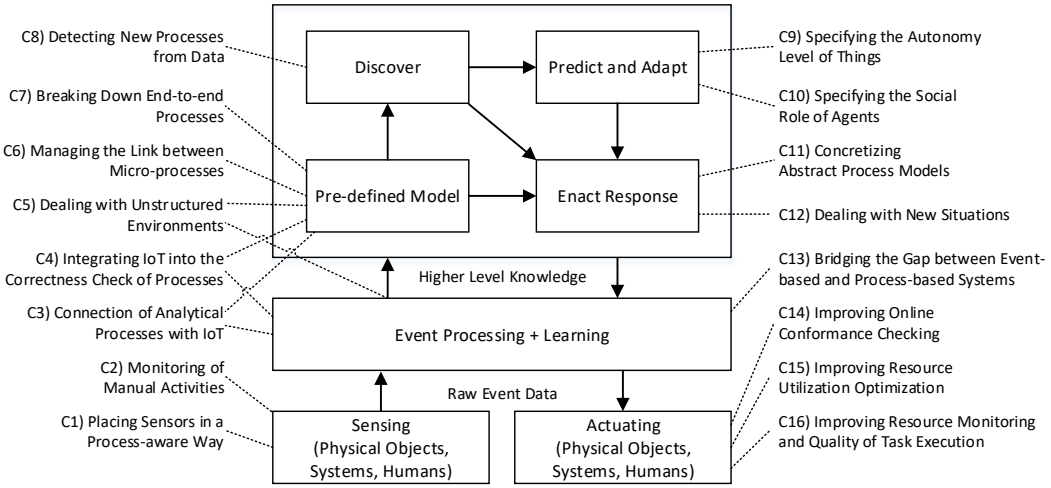


Figure 2.16.: High-level Overview of Challenges Showing the Interaction between IoT and BPM from [JKM+17].

Before exploiting the comprehensive list of advantages presented in Section 2.5.2, various obstacles and issues have to be resolved that come with the introduction of workflow technologies in CPS. The application of BPM technologies in the context of CPS and IoT poses a new research field that raises various novel research challenges. Several works discuss this topic and with that the new challenges in detail [LMM15, CSB15, CSB16, MBBF17, GGAAPE+11, MRH15, KP15]. The work by Janiesch et al. presents a good high-level overview of 16 challenges related to the interaction between IoT and BPM that are partially relevant for our work (cf. Figure 2.16) [JKM+17]. This overview relates to the phases of the BPM lifecycle (cf. Section 2.3.2) extended with the lower layer processing and learning of raw data from sensing and actuating of entities in IoT and CPS. From this connection, 16 challenges are identified that discussed in detail in [JKM+17]. When deriving requirements for our work, we will refer to the challenges from this list.

The more general challenges that we want to address with this thesis refer to the establishment of workflow technologies in the domains of IoT and CPS. These systems consist of a large number of heterogeneous entities comprising various sensors for environmental factors (e. g., light levels, temperature, humidity) and actuators for controlling appliances (e. g., light switches, thermostats, service robots), smart objects equipped with sensing technologies (e. g., RFID and NFC) as well as virtual sensors and software services. All these CPS entities interact with each other, with the physical environment and physical objects, with the virtual world, and most importantly with the users. We aim at developing a workflow notation and execution system to describe and enact interactions among these entities in the physical and virtual worlds on the business process level (PACPS). With the physical world being more unreliable and producing more unanticipated situations, the WfMS has to be resilient and self-adaptive to deal with errors and unexpected behaviour. To achieve these properties, the effects of the workflow executions in CPS have to be described

as part of the processes and verified during process execution. We aim at creating a link between the workflow executions and the corresponding effects on physical objects and the physical environment, and vice versa.

2.6. Requirements

Based on the previous findings regarding the properties P1–P10 of CPS (cf. Section 2.4.6), process levels in PACPS (cf. Section 2.5.1), challenges C1–C16 of bringing together IoT and BPM (cf. Section 2.5.3) as well as related research, we derive a set of high-level requirements that this PhD thesis will cover. The requirements are classified into two categories: 1) related to the basic *CPS Workflow Notation and Engine*, and 2) to *Self-management for Cyber-physical Resilience*. The lists of requirements and challenges are by far not exhaustive as this thesis is one of the first research works in the area of CPS and BPM. However, it discusses the basic requirements necessary to establish a dedicated layer for resilient workflows in CPS towards PACPS and therefore to exploit the advantages workflows may offer for programming and controlling CPS (cf. Section 2.5.2). For each requirement, we will denote its relation to the corresponding CPS properties (P1–P10) from Section 2.4.6 and challenges (C1–C16) from Section 2.5.3 as well as references to corresponding external sources. Despite the main application domain of this thesis being the *Smart Home* as an example of CPS, most related works discuss CPS-related properties and challenges with respect to other domains (e. g., smart factories, smart hospitals, automotive). However, we are aiming at deriving a general set of requirements for implementing CPS workflows based on the general properties of CPS and their main actors with a focus on the interactions of the virtual entities with the physical world and vice versa. These general requirements and their associated solutions can then also be applied to the smart home domain.

2.6.1. CPS Workflow Notation and Engine

The following requirements refer to features that a CPS WfMS has to support to serve as basic WfMS for implementing CPS workflows in the context of this thesis. This also comprises the respective workflow notation to specify the CPS workflows.

R1: Abstraction and Processing of Complex Sensor Events With CPS consisting of a possibly large number of sensors and other event sources, the basic CPS WfMS has to support the processing and abstractions of complex sensor events on different levels of granularity from low-level sensor events to aggregated higher level events. This aspect also includes the correlation of the appearance of specific sensor events with the execution of related workflow instances and vice versa (*Instance Correlation* [Wom11b]).

Refers to: P1, P2, C1, C2, C3, C13, [BBDC+15, Tal08, OHG17]

R2: Integration and Dynamic Selection of Resources Due to resource-constraints and context-dependant behaviour of heterogeneous devices in CPS, the CPS WfMS has to support the integration and dynamic selection of process resources at runtime.

Refers to: P4, P6, C5, C15, [BCG12, CDB+12, SDFGB09, CSB16, SGCG18]

R3: Ubiquitous Interaction with Humans As humans are important first class entities who interact with processes in the form of process resources or who manage the process executions, the CPS WfMS has to support the integration of humans into workflows and provide means for ubiquitous interactions with the WfMS.

Refers to: P6, P7, C2, C16, [BAJ17, BCG12, Wei91, SECP13]

R4: Distributed Process Execution With hierarchical structures and the large number of devices that make up CPS as well as an increasing demand for scalability, autonomy and decentralization, the CPS WfMS has to support the distributed–and possibly offline–execution of complex processes in hierarchical networks.

Refers to: P2, P4, P10, C6, C7, C15, [CDB+12, Bro13, MMG08, PRS+13]

2.6.2. Self-management for Cyber-physical Resilience

The following requirements refer to advanced features a CPS WfMS has to support to provide a resilient WfMS that is capable of maintaining a consistent state during the cyber-physical interactions and dealing with unanticipated situations and errors in CPS. The last requirement refers to the extension of existing WfMSes to also support these features, which will be investigated in the context of this thesis, too.

R5: Cyber-physical Synchronization Due to the inherent nature of CPS, interactions between the virtual and the physical world take place very frequently. A digital representation of physical objects, processes and context factors has to be available to the CPS WfMS and constantly synchronized through a feedback loop-based mechanism to keep a consistent state of the physical and virtual worlds.

Refers to: P1, P5, P9, C3, C4, C14 [Lee08, LBK15, CSB16]

R6: Handling of Cyber-physical Errors With the previous requirement, there is also a need to react automatically to inconsistencies, deviations, imprecisions and other errors that may have occurred during the execution of CPS workflows. The CPS WfMS has to be supported with this task by a dedicated software component.

Refers to: P3, P5, P8, P9, C4, C8, C9, C12, [CBF+16, BCG12, HWS+16]

R7: Self-management Capabilities Due to the highly context-dependant and dynamic behaviour of CPS and building on the requirement regarding the automated handling of cyber-physical errors, the CPS WfMS has to also be able to react autonomously to other undesired situations and errors based on specified constraints related to context factors, performance indicators or service quality levels.

Refers to: P6, P8, P9, P10, C5, C9, C12, C14, C15, [GGBG13, MSW16, Kou11, OCEP13]

R8: Retrofitting of Workflow Management Systems As the capability of self-management is an essential feature of a CPS WfMS and many (non-CPS) WfMSes already exist in industrial and academic contexts, a retrofitting framework and process have to be derived to also equip these existing WfMSes with the capability of self-management to enable cyber-physical resilience and other self-* properties.

Refers to: P1, P5, P6, P8, P9, P10, C5, C9, C12, C14, C15, [GGBG13, MSW16, Kou11, OCEP13, ABD+16]

2.6.3. Research Questions

From the elaborations in this chapter, we derive the central automation hypothesis for this PhD thesis:

> Workflow technologies can be used to
> - facilitate the linking of components,
> - increase the level of automation,
> - and enable resilient autonomous processes
>
> in Cyber-physical Systems.

With regard to that hypothesis, we will investigate the following research questions related to the corresponding requirements *R1–R8* in the course of this thesis:

Q1) How to model workflows in CPS? (*R1, R2, R3, R4*)

Q2) How to design and implement a CPS WfMS? (*R1, R2, R3, R4*)

Q3) How to synchronize virtual and physical world processes? (*R5*)

Q4) How to add self-* capabilities to a CPS WfMS? (*R6, R7*)

Q5) How to retrofit existing workflow systems? (*R8*)

The research questions can be classified according to the simplified BPM lifecycle described in [VDA13]. Question *Q1* relates to the *(Re)design* phase; question *Q2* relates to the *Implement/configure* phase; questions *Q3* and *Q4* relate to the *Run and adjust* phase; and question *Q5* relates to all three phases. These research questions serve as the basis for discussing related research in the next chapter. We structure this investigation into five categories derived from the research questions:

- Modelling of CPS Workflows (*Q1*),
- CPS Workflow Systems (*Q2*),
- Cyber-physical Synchronization (*Q3*),
- Self-* for BPM Systems (*Q4*),
- Retrofitting Frameworks (*Q5*).

3. Related Work

"Science never solves a problem
without creating ten more."

George Bernard Shaw

3.1. Introduction

After identifying the important requirements and research questions that will form the scope of this thesis, we discuss related research regarding the application of BPM technologies for CPS. We will investigate and evaluate approaches discussing the topic and various aspects of applying BPM technologies in the context of CPS and IoT. The goal is on the one hand, to identify research gaps, and on the other hand, concepts that can be applied to realize a WfMS for CPS on the modelling, implementation and runtime level fulfilling the requirements that form the scope and theme of this thesis. First, we briefly evaluate some well-known and established existing WfMSes from industry and academia with respect to the requirements identified in Section 2.6. Following, a detailed discussion of related research with respect to these requirements is given. This discussion is structured in the following categories, which can be related to the phases of the simplified BPM lifecycle [VDA13]: Modelling of CPS Workflows (*Design*), CPS Workflow Systems (*Implement/configure*), Cyber-physical Synchronization (*Run and adjust*), Self-* for BPM Systems (*Run and adjust, Redesign*), and Retrofitting Frameworks. The results of this evaluation are then used as basis for our own concepts that will be described in the following chapters.

The evaluation of related research in this chapter is based on a comprehensive literature study regarding the topics *Business Process Management*, *Workflows*, *Cyber-physical Systems* and *Internet of Things*. The study is complemented by various literature surveys on these topics [CSB16, SJV+15, MSW16, TMS+12, SLI08, SWYS11, AIM10, VDA13, GBMP13, MSDPC12, MBBF17, Bor14].

3.2. Existing BPM Systems in Industry and Academia

BPM technologies have been around for many years and with that, a lot of WfMSes have been developed. In this section, we briefly evaluate some of the most well-known systems that are used in industry and academia. Table 3.1 gives an overview of 12 WfMSes and an evaluation with respect to their fulfilment of the requirements *R1–R8*. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective system: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

Table 3.1.: Evaluation of Existing WfMSes with respect to Requirements.

WfMS \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
Activti	o	-	+	o	-	-	-	-
jBPM	+	o	+	o	-	-	-	-
Apache ODE	o	-	o	-	-	-	-	-
YAWL	-	+	o	o	-	-	-	-
Camunda	o	-	o	o	-	-	-	-
Bonita	o	-	o	o	-	-	-	-
MS WWF	-	-	o	-	-	-	-	-
ARIS	-	-	o	o	-	-	-	-
SAP	-	-	o	-	-	-	-	-
Bizagi	+	-	+	o	-	-	-	-
IBM	o	o	o	-	-	-	-	-
IFTTT	o	o	+	-	-	-	-	-

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

Activiti Activiti[1] is an open source WfMS implemented in Java. It supports BPMN 2.0 processes and provides an execution engine and model repository as well as graphical modelling and monitoring tools. Activiti does not support the integration and dynamic selection of specific CPS components. Simple sensors could be integrated using BPMN's event specifications. The WfMS provides web-based tools for modelling, monitoring and collaboration to facilitate human interactions. A basic form of distributed process execution could be reached using BPMN's pools and lanes. There is no support of aspects regarding *Cyber-physical Resilience* (Requirements *R4–R8*).

jBPM jBPM[2] is a Java-based open source WfMS, which supports BPMN 2.0, WS-BPEL and other workflow notations. It allows for coupling with additional jBoss tools, e. g., to enable Complex Event Processing (CEP) of sensor streams. Dynamic processes and services can be achieved by various mechanisms of underspecification and the integration with Open Services Gateway initiative (OSGi) services. It also features graphical tools and *Human Tasks*. There is no explicit support of sensors or actuators, nor of requirements related to cyber-physical resilience.

Apache ODE Apache ODE[3] (Orchestration Director Engine) is a Java-based WfMS with a focus on service orchestrations using WS-BPEL for executable process definitions. It provides standard workflow functionality including service calls to proprietary and SOAP-based web services, simple events and human tasks. Special CPS properties are not addressed by Apache ODE.

[1] `https://www.activiti.org/`
[2] `https://www.jbpm.org/`
[3] `http://ode.apache.org/`

YAWL The YAWL[4] engine is a Java-based workflow system that relies on YAWL as basic workflow notation for processes (cf. Section 2.3.3). It allows for modelling of complex workflows involving data transformations and service invocations based on Petri nets. Dynamic processes and services can be implemented using *Worklets* [ATHEVDA06]. The Petri net based approach could also be used to model workflows in distributed systems. Again, there is no explicit support for CPS specific components and resilience.

Camunda Camunda BPM[5] is a BPM platform mostly focussed on software development processes. It uses BPMN 2.0 as basic workflow notation. The Java-based execution engine features integration with Spring[6], SOAP and REST service connectors as well as *Human Workflow Management*. BPMN's events and pools/lanes could be used to realize sensor interactions and distributed processes. Other CPS aspects are not supported.

Bonita Bonita BPM[7] is a Java-based WfMS, which also relies on the BPMN 2.0 standard for describing business processes. It includes a workflow designer, web portal to manage processes, an execution engine and various connectors for third party applications. The tooling for interacting with the WfMS is comprehensive, events and distributed execution could be realized similarly to the other BPMN 2.0-based engines. The specific CPS requirement cannot be fulfilled.

MS WWF The Microsoft Windows Workflow Foundation[8] is a more technical approach aimed at developers for implementing automated workflows among Microsoft products. It uses a proprietary format and execution engine to enact the processes. Besides some graphical user interfaces to model and interact with the workflows, there is no support of relevant aspects with respect to our investigations.

ARIS The ARIS Platform[9] is an extensive platform and tool suite for modelling, managing, analysing and executing business processes. The products support a wide variety of modelling and web techniques (including BPMN, WS-BPEL and UML) and are applied in various domains of software engineering and BPM. A native support of CPS or IoT related aspects cannot be found, though.

SAP The SAP Business Workflows are part of the SAP applications[10] for managing business processes. They are tightly integrated with the SAP software suites and their business objects. Various tools for workflow modelling, monitoring and managing exist. The properties of CPS are not supported.

[4] http://www.yawlfoundation.org/
[5] https://camunda.com/
[6] http://spring.io/projects/spring-framework
[7] https://www.bonitasoft.com/
[8] https://msdn.microsoft.com/en-us/library/jj684582.aspx
[9] http://www2.softwareag.com/corporate/products/aris_alfabet/bpa/default.aspx
[10] https://www.sap.com/germany/products.html

Bizagi Bizagi[11] Business Process Management provides various mature software tools for workflow management. The Bizagi Modeler and Studio support BPMN 2.0 specifications and execution. The focus is on virtual business processes in various organisational and commercial domains. Few CPS features can be supported by adapting BPMN's event and pool/lane mechanisms.

IBM IBM's Node-RED[12] platform allows for a flow-based programming for the IoT. The platform is based on Node.js[13] and enables the wiring of integrated IoT devices to form workflows. Simple sensors and actuators can be part of these flows. A formal workflow notation is not available. Complementary to that, IBM BlueWorks Live[14] is a Cloud-based business process management platform that does not cover CPS aspects, though.

IFTTT If This Then That[15] is a simple end-user platform to create automated flows between web and IoT services. Sensor events can be used to trigger actuators and services to execute specified actions (ECA rules). The focus of this platform is on the flow-based orchestration of web services, e. g., to integrate social media applications. There is no formalism underlying the workflow definitions and no support of the CPS-related requirements.

Conclusion

From the investigations related to the fulfilment of requirements *R1–R8* by existing WfMSes, we conclude that there is basically no native support of CPS-related aspects by current BPMSes used in industry. Some of the necessary features could be implemented by adapting and extending already available mechanisms (e. g., CEP for sensor processing or *Human Tasks* for interactions). Almost all of the presented BPM systems put their focus on managing (virtual) business processes–either inter-organizational or intra-organisational–that describe the flow of interactions among humans, goods, organisational units and companies. Despite the presented systems being successful in these areas, they cannot be applied within CPS due to a lack of support of basic and advanced features that are required of a CPS WfMS. Besides not fulfilling most of the identified requirements, some of the BPM systems are rather bloated and rely on heavy-weight SOA technologies. CPS and IoT environments require fast and light-weight software components and communications due to limited resources and constraint environments [GIM11].

We have seen that there is a wide spectrum of already existing WfMSes, which are all capable of invoking external services and software components. With requirement *R8*, we will discuss possible extensions and other means of adapting these systems to add some support of CPS workflows and self-management to their functionalities. Based on the discrepancies determined by the investigation of the existing WfMSes, we will conduct a more detailed review of research from academia to evaluate the suitability of related approaches w. r. t. requirements *R1–R8*.

[11] `https://www.bizagi.com`
[12] `https://nodered.org/`
[13] `https://nodejs.org/en/`
[14] `https://www.blueworkslive.com/`
[15] `https://ifttt.com/`

Listing 3.1: Example of the *When–Then* Extension for WS-BPEL.

```
<iotx:when name=?temperatureCondition?>
 <bpel:condition>$temperatureVar > 35</bpel:condition>
 <sequence name=?test?>
  <empty name=?empty1?></empty>
  <empty name=?empty2?></empty>
 </sequence>
</iotx:when>
```

3.3. Modelling of CPS Workflows

The idea of using BPM technologies in the context of IoT and CPS to describe and execute, and thereby automate repetitive tasks has gained more and more interest over the last years. First, we will investigate approaches with respect to requirements *R1–R4* that refer to the modelling of workflows for CPS and IoT. Common workflow notations (e. g., WS-BPEL, BPMN 2.0 and YAWL) do not natively support the specification of sensor-related events and actuator operations. They also do not feature the modelling of physical objects/things and their context properties as well as dynamic behaviour and resource constraints. Various works investigate the applicability of existing workflow languages to model (business) processes in the new domains of IoT and CPS [DMC14, GEPF11, MRH15, BBDC+15, SSOK13, TSD+12, Mez16, YBSD16, GKGK16, MD17, BDGP17]. All of these investigations come to the conclusion that the expressiveness of established BPM languages is not sufficient to model CPS and IoT-specific properties and behaviour regarding the workflow-based interaction with sensors, actuators, humans and the physical world in general. They propose extensions for various modelling notations to represent these new entities as resources, new participants in workflows and other aspects.

In [DMC14] Domingos et al. propose an extension of the standard WS-BPEL language to also support context variables and communication paradigms regarding the IoT. Their extensions relate to sensor values as new context variables and the interaction of the processes with these sensors in a *request/reply* or *publish/subscribe* manner. On top of that, *when-then* conditions can be defined to enable simple ECA rule specification and event-driven behaviour. Listing 3.1 shows an exemplary *when–then* construct related to a specific temperature and executing one of two activities depending on the temperature value. The authors describe a mapping process for the extension to standard WS-BPEL workflows to maintain compatibility among WfMSes. Standard WS-BPEL mechanisms and extensions can still be used (e. g., for dynamic services or human interaction).

Kefalakis et al. present the *APDL* process definition language for complex RFID-based sensor solutions and processes [KSKP11]. The XML-based Domain-specific Language (DSL) is designed to model business processes involving RFID sensors as key data and event sources in production and supply chain management. The *APDL* specification is an extension of the XML Process Definition Language (XPDL) for specifying more technical processes. It is accompanied by a comprehensive editor.

The majority of extensions proposed in related work refer to BPMN as basic notation as it is currently viewed as de facto standard for modelling business processes.

The applicability of BPMN in the context of Wireless Sensor Networks (WSNs) to describe IoT-related processes is evaluated and proposed for building automation in [CK11, TSD+12, SSOK13]. With *BPMN4WSN* Tranquilini et al. [TSD+12] and Sungur et al. [SSOK13, MPO+17] describe extensions of BPMN to model and program business processes related to sensor networks. Figure 3.1 shows the proposed extensions with respect to the BPMN metamodel. The central class is the *WSNTask* as specialization of the standard BPMN *ServiceTask*. A *WSNTask* represents a specific action to be executed in WSNs–either a sensing task, an actuating task or an intermediary operation. All of which can be active command actions or event-driven tasks. *WSNPerformers* can be specified to define static or dynamic resources (nodes) that are responsible for executing the specific task. The execution logic maybe distributed among several performers. A *WSNTask* refers to specific *WSNOperation*s, which are invoked to execute the task.

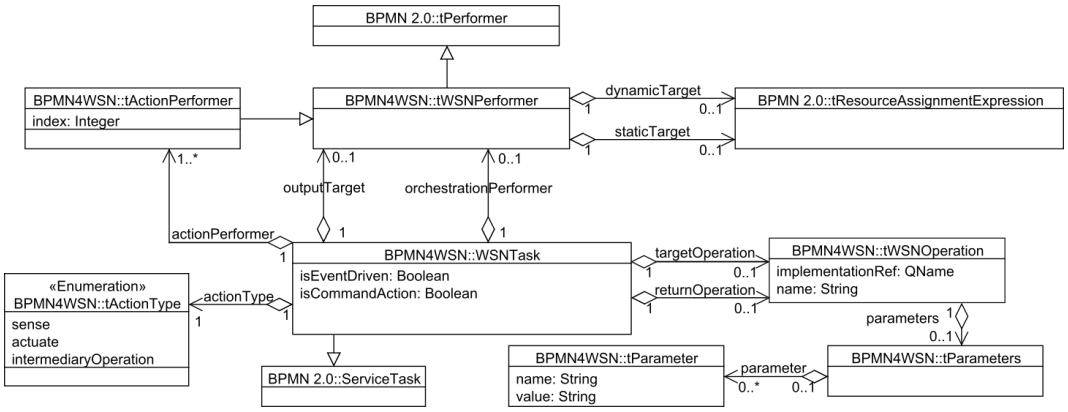


Figure 3.1.: *BPMN4WSN* Class Diagram from [SSOK13].

As the modelling of individual sensors and event sources is often not feasible for complex IoT and CPS setups, Baumgraß et al. show the application of CEP for BPMN processes in logistics scenarios [BBDC+15]. The extension of a BPMN task with an Event Processing Language (EPL) query allows for defining specific patterns in complex event streams that will trigger certain actions to be performed. The authors also propose a corresponding system architecture for event-driven process execution systems, which will be explained in more detail in Section 3.4. Another approach for modelling and execution of event stream processing in business processes is proposed by Appel et al. in [AKF+14]. They introduce Event Stream Processing Units (*SPU*s) as encapsulations of application logic for stream processing and show how to execute these units with respect to EPCs and BPMN.

The integration of IoT devices as business process resources based on BPMN 2.0 is investigated by Meyer et al. in [MRM13]. They base their approach on the linking of the IoT devices and their native software services based on the *IoT Reference Model* (cf. Section 2.4.4) with BPMN resources. IoT devices are represented by extensions of BPMN *Lanes* in process pools. The device's native service is exposed as process performer. The dynamic assignment of an IoT service or a required native service to execute a certain task at runtime can be achieved by using a specific *Expression*, which is part of an IoT parameter definition or an explicit IoT assignment specification. As an extension of these concepts, the authors discuss various ways of

representing *Things* within business processes in [MRH15]. They propose to represent physical entities (*Things*) as *Participants* in BPMN-based processes, i. e., as pools and lanes with no active tasks.

The works of Yousfi et al. discuss BPMN extensions for modelling business processes and interactions in the context of ubiquitous systems, i. e., with a stronger focus on interacting with users via interactive devices [YBSD16, YHBW17]. The authors propose the introduction of new task types (*Sensor Task, Reader Task, Image Task, Collector Task*) regarding the use of new smart devices to gather data from smart objects (e. g., by using microphones, cameras or RFID readers). Along with that, new event types (*Sensor Event, Reader Event, Image Event, Audio Event, Collector Event*) are introduced to the business processes to indicate the capturing of new data from a smart object after executing one of the corresponding tasks. In [YHBW17] Yousfi et al. extend and apply these concepts to describe new patterns regarding additional characteristics and new paradigms of ubiquitous systems, namely automatic identification and data capture, context awareness, augmented reality, sustainability, and ambient intelligence.

Figure 3.2 presents an overview of the previously discussed works with examples for IoT-driven business process notations from [CSB16]. The figure shows special tasks for sensing and actuating marked with the respective icons as proposed by Sungur et al. [SSOK13]. The exemplary IoT process also contains dedicated pools and lanes for the IoT system and IoT devices as discussed by Meyer et al. [MRM13] as well as their proposal of representing physical entities (here: *Milk*) as passive participants in a business process [MRH15]. Additionally, the process shows the emission of an event by a smart actuator ([YBSD16]) and the execution of a sensor stream task by an SPU [AKF+14].

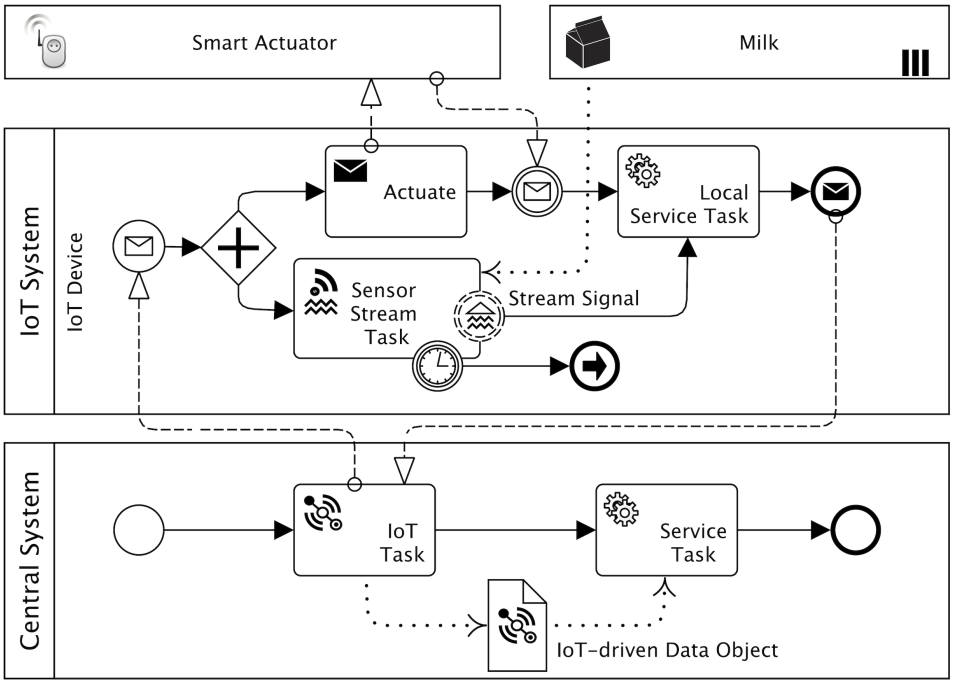


Figure 3.2.: Examples for IoT-driven Business Process Notations from [CSB16].

With *BPMN4CPS*, Graja et al. present an extension of the BPMN 2.0 metamodel to model relevant CPS aspects in business processes [GKGK16]. Besides a separation of pools for physical and cyber activities, a *Controller* pool is responsible for orchestrating the overall process. Regarding the types of activities, the authors distinguish between *Cyber Tasks* (*Embedded Service Task, Web Service Task, Cloud Service Task*) and *Physical Tasks* (*Actuator's Task, Sensor's Task*) as extensions of BPMN *Service Tasks*. Special CPS devices including their real world parameters and physical entities can be defined as subclasses of BPMN *Performers*.

In [BDGP17] Bocciarelli et al. present an extended version of their *PyBPMN* modelling language for CPPS. The *PyBPMN* extension of the BPMN language enables the definition of workload, performance, reliability and other general resource management properties [BD11]. The new *PyCPS* extensions aim at defining more detailed CPS-specific properties of the process resources (e. g., components, control units, sensor and actuator units and their communication interfaces).

Martins and Domingos discuss the modelling of IoT behaviour within business processes using only standard compliant BPMN 2.0 modelling elements in [MD17]. Their approach is based on defining specific tasks of IoT devices as resources in dedicated pools. The resulting BPMN processes are translated to the *Callas* sensor programming language [LM16]. The corresponding bytecode is then deployed and executed on the respective IoT devices.

A semantic framework for modelling the management of IoT resources in business processes is proposed by Suri et al. in [SGCG17]. The authors present extensions of an ontological representation of core BPMN classes [YCZM07] to model human and non-human process resources, e. g., various types of IoT devices (sensors, actuators, tags). Special IoT-related attributes, constraints and properties of these resources can also be defined with the help of the ontologies. These properties can then be used to find replacing resources in case of errors or conflicts at runtime.

Other related works discuss the usage and modelling of IoT devices in the context of smart business processes [GGAAPE+11, FVH18]; the collaboration of multiple IoT devices within a *process-aware IoT community* to solve specific goals in a process-driven way [KAK16]; or the extension of BPMN 2.0 with resource and context constraints to enable business process executions on mobile devices [PRS+13]. In [SWC+18] the authors describe a new approach for integrating semantic context information into Petri net based business processes in the context of IoT.

Conclusion

Table 3.2 presents an overview and evaluation of work related to the modelling of CPS workflows that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

Most works identified a need to extend existing BPM notations to represent specific CPS and IoT devices and behaviour as well as physical world entities and properties as part of a business process. Sensors and actuators are usually integrated as process resources in a service-based manner (cf. Section 2.3.5). However, they are addressed rather statically, which limits the dynamic selection of suitable resources at runtime (Requirement *R2*). Most works propose the modelling of in-

Table 3.2.: Evaluation of Related CPS Workflow Modelling Approaches with respect to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[DMC14]	+	o	o	-	-	-	-	-
[KSKP11]	+	-	o	-	-	-	-	-
[SSOK13]	+	+	o	o	-	-	-	-
[BBDC+15]	++	o	o	-	-	-	-	-
[AKF+14]	++	o	o	-	-	-	-	-
[MRM13]	+	+	o	-	-	-	-	-
[YBSD16]	+	o	++	-	-	-	-	-
[GKGK16]	o	+	o	-	-	-	-	-
[BDGP17]	+	o	o	-	-	-	-	-
[MD17]	o	o	o	-	-	-	-	-
[SGCG17]	o	+	+	-	-	-	-	-

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

dividual sensors and sensor tasks, which are communicated with via active service calls or publish/subscribe mechanisms. For larger networks of various types of sensors in CPS, the approach of addressing sensors individually becomes infeasible, which is why CEP was introduced as a solution to process more complex streams of sensor events (Requirement *R1*). The aspects of human interaction (Requirement *R3*) and distributed processes (Requirement *R4*) are addressed only to a very limited extend by few works. All of the proposed extensions to existing more or less standardised workflow notations also lead to the implementation of proprietary extensions to existing or completely new proprietary WfMSes. The more CPS-related requirements of synchronization, error handling, self-management and retrofitting (Requirements *R5–R8*) are not addressed explicitly by any of the investigated works.

3.4. CPS Workflow Systems

Following the modelling of CPS workflows (*Design Phase*), the implementation and execution of the particular workflows by a CPS workflow system (*Implementation/Configuration Phase*) need to be investigated. Various works present approaches for WS-BPEL-based, BPMN-based or proprietary WfMSes that address specific properties of CPS and IoT environments for different contexts and domains [GEPF11, CS11, JDK15, BCD+15, TSD+12, HHGR06, SHH+14, PRBA15, JROK11, MCS16, MM05, MPMR16]. A comprehensive overview of BPM architectures discussing WfMSes with respect to reference architectures, adaptive processes and service composition can be found in [Wes12]. In this section, we discuss WfMSes for CPS with a focus on realizing the basic set of requirements *R1–R4*. More advanced approaches that also investigate the self-management aspects including

autonomic error handling (*self-healing*) and cyber-physical interactions (synchronization) are presented in Section 3.6.

Glombitza et al. present an approach for using WS-BPEL to realize business processes for the IoT in [GEPF11]. They propose a SOA-based architecture to transform and execute WS-BPEL processes on sensor nodes and WSNs acting as runtime environments for web services and application logic as defined in the respective WS-BPEL processes. The goal is to use resource-constraint IoT devices to enact heavy-weight business processes after being transformed to the respective target platforms. The authors do not provide any IoT or CPS-specific WS-BPEL extensions to model processes. The *Flogo*[16] project follows a similar approach of modelling workflows involving micro-services for IoT and transforming these workflows to executable code for the respective edge devices (cf. Section 2.3.5).

With *ERWF*, Chen and Shih developed an "Embedded Real-Time Workflow Engine for User-centric Cyber-physical Systems" [CS11]. This engine uses proprietary workflow scripts to describe and execute predefined activities as processes on a humanoid robot serving as an example of an embedded system combined with actuators and sensors. The authors define a formal model for the execution times and probabilities of workflow activities in order to enable real-time capabilities of the workflow engine. Figure 3.3 presents an overview of ERWF's system architecture. *Workflow Scripts* are instantiated by the *Workflow Manager & Processor*. The *Scheduler* is responsible for assigning *Threads* to workflow tasks, which are then dispatched to execute the workflow instances–possibly with real-time behaviour. Specific *Device Drivers* are used to communicate with the robot's built-in sensors and actuators.

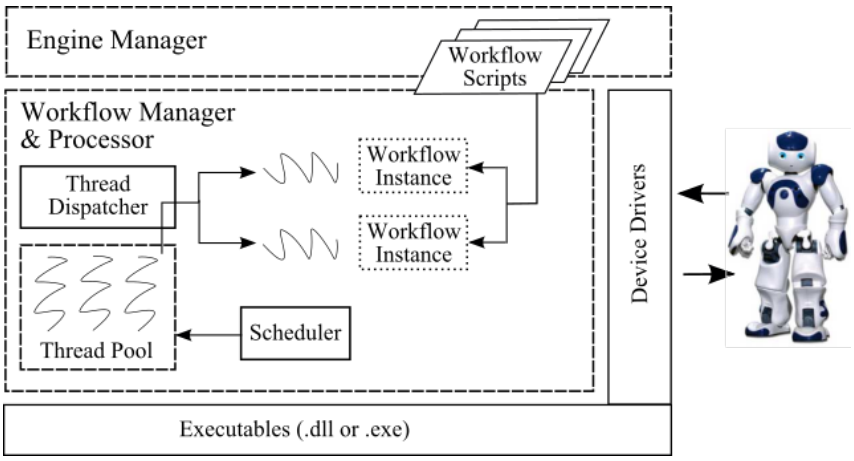


Figure 3.3.: System Architecture of the ERFW System from [CS11].

In [JDK15] Juhász et al. present the *Rea* framework for programming workflows for CPS. It follows a task-oriented programming approach for implementing workflows on a hardware-related level consisting of loosely-coupled tasks, combinators (sequences, parallels, controllers, pipes) and constraints. The *Rea* language is a DSL founded on an extended version of the Erlang programming language. The workflow system runs on the corresponding IoT devices and interacts with its sen-

[16] `http://www.flogo.io/`

sors and actuators. The framework provides means for failure detection and handling as well as interaction with the workflows via a simple client application.

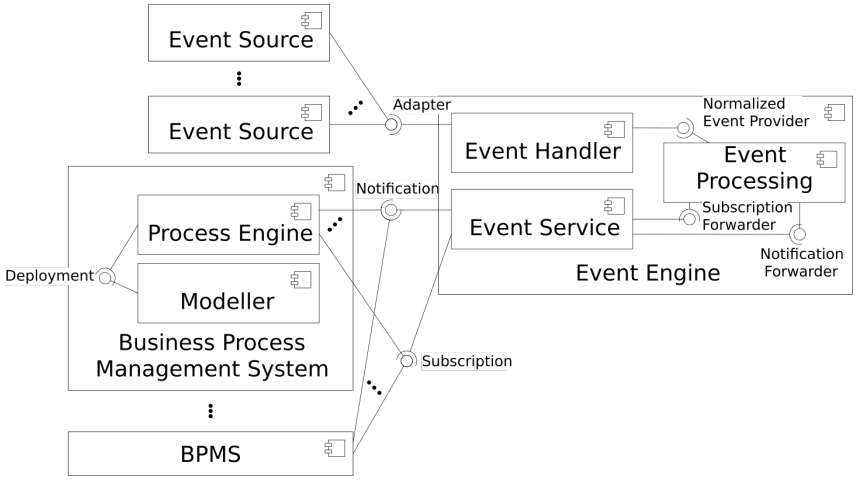


Figure 3.4.: High-level Architecture for Real-time Monitoring of Business Processes through CEP from [BBDC+15].

The implementation of event-driven process applications in the domains of logistics and IoT was proposed in several works by Herzberg et al. [HMW13], Baumgraß et al. [BBDC+15, BCD+15] and Mandal et al. [MHW17]. The authors propose to use CEP to evaluate complex event streams from various sensor sources based on EPL patterns defined in the respective process models. Figure 3.4 shows a high-level architecture for real-time monitoring of business processes using CEP as presented in [BBDC+15]. The *Process Engine* subscribes to an *Event Service*, which is part of the *Event Engine*. Various *Event Sources* are connected to the *Event Processing* component via *Event Handlers*. Once a significant pattern is detected, the event processing component emits a notification, which is forwarded to all subscribed process engines through the respective event services. Another approach applying CEP in the context of business processes to analyse sensor streams is presented by Jung et al. in [JROK11]. Event Stream Processing Units (*SPU*s) proposed by Appel et al. for event processing tasks in business processes [AKF+14] were already introduced in Section 3.3.

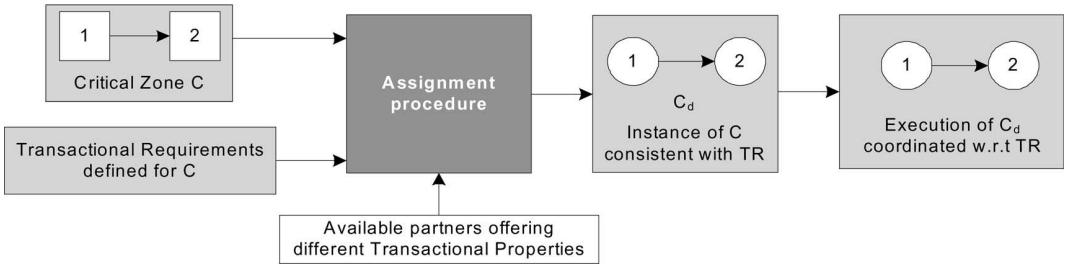


Figure 3.5.: Methodology for Implementing Transactional Workflows from [MMG08].

Pervasive workflows supporting long running transaction in distributed environments are discussed in detail by Montagut et al. in [MM05, MMG08]. The authors propose a fully decentralized workflow engine based on web services, the dynamic assignment of business partners to workflow tasks, and WS-BPEL as underlying workflow language. Along with that workflow system, a transactional protocol for workflow tasks was developed to deal with failures occurring in the decentralized infrastructure of process resources. Figure 3.5 shows their proposed methodology for implementing transactions for workflows. A *Critical Zone* C is defined for a part of a workflow along with *Transactional Requirements* (TR) for C. During the *Assignment procedure*, available partners are evaluated with respect to their offered transactional properties. In case a partner fulfils the requirements, an *Instance* C_d of the respective workflow activities in C is created and executed with respect to the transactional requirements.

Complementary to the previous approaches, the *ADEPT* WfMS also supports the distributed execution of workflows based on *instance migration* as well as ad-hoc changes to the underlying processes [RRD03, DR09]. Using the ADEPT system as basis, Müller et al. propose the *AgentWork* WfMS supporting rule-based workflow adaptations [MGR04]. The ADEPT system and the corresponding *AristaFlow* WfMS [DRRM$^+$09] for flexible, adaptive and highly scalable business processes is currently being used in various IoT-related projects to implement flexible and mobile production processes [PRBA15].

Worklets proposed by Adams et al. represent self-contained subprocesses in SOAs flexibly selected at runtime depending on the specific workflow tasks and their context [ATHEVDA06]. *Exlets* are specializations of these dynamic subprocesses specifically designed to deal with exceptions during process execution [ATHVDAE07]. The AristaFlow system also implements robust and flexible error handling mechanisms [LRD10].

Dar et al. present in [DTB$^+$15] a process-based and resource-oriented integration architecture for the IoT. Their approach is set in smart home and AAL settings to provide users with assistance. The corresponding processes are described using BPMN. The framework supports dynamic service discovery and replacement based on web service descriptions and registries; event-based communication based on publish/subscribe mechanisms; and distributed process execution based on BPMN's choreography capabilities–also on mobile IoT devices.

Another approach for mobile and distributed business process in the context of IoT is proposed by Pryss et al. [PTKR10, PRBA15]. Their focus is on designing a flexible light-weight workflow system to be used on mobile devices. The system uses context information and additional sensor data to provide a mobile task execution environment in a nursing home scenario. A 3D augmented reality application can be used to model, configure and visualise the corresponding workflows. With *Presto* and *Parkour*, Giner et al. discuss similar approaches of mobile workflow support in the IoT [GCFP10]. Their focus is on supporting human workers with executing tasks within business processes and on implicit interactions for pervasive workflows in subsequent work [GCFP11]. An approach for integrating external context information from wearables into business processes in a production context can be found in [SAEJ18]. Peng et al. also present a mobile WfMS in a Cloud computing setting [PRS$^+$13]. The central process executor and coordinator is deployed on a Cloud

server, which distributes subprocesses and tasks to a mobile process engine. The mobile engine is able to incorporate context information from various sensors and to dynamically invoke required services in the Cloud as part of the activity execution.

In [MCS16] Mass et al. describe a device-to-device-based BPMS for industrial IoT. The BPMS enables decentralized process execution, also with unreliable IoT devices. A *Migration Module*, which is part of the *Process Executor* is capable of migrating a process instance from one node to another by creating a snapshot of its current state and transferring this instance to a new process executor.

Other related works comprise a general context-aware and workflow-based framework for pervasive environments, which can be used with various CEP and workflow engines [AE14]; a WS-BPEL-based workflow execution engine for mobile devices [HHGR06]; a Cloud Process Execution Engine (*CPEE*) [MRM14]; and a conceptual framework for process-based Cloud manufacturing [SHH+14]. The concept of *Osmotic Computing* for distributing simple process tasks between Cloud and Edge in an IoT context is presented in [NND+17]. An approach describing the smart configuration of smart environments based on the semantic description of IoT resources and with that, the goal-oriented derivation and configuration of processes in the context of smart homes and smart factories can be found in [MVKM16, MPMR16].

Conclusion

Table 3.3.: Evaluation of Related CPS Workflow System Approaches with respect to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[GEPF11]	o	o	-	-	-	-	-	-
[CS11]	+	o	-	-	-	-	-	-
[JDK15]	o	-	o	-	-	-	-	-
[BBDC+15]	++	o	o	-	-	-	-	-
[MMG08]	-	++	-	++	-	-	-	-
[DTB+15]	+	+	o	+	-	-	+	-
[PRBA15]	+	o	+	+	-	-	-	-
[GCFP10]	+	o	++	o	-	-	-	-
[PRS+13]	+	+	o	+	-	-	-	-
[MCS16]	+	+	o	++	-	-	-	-

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

Table 3.3 presents an overview and evaluation of work related to the execution/implementation of CPS workflows that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

Within the investigations of related work regarding CPS WfMSes we found approaches discussing several relevant aspects. Event-driven architectures as well as CEP were identified as important technologies to cope with sensor and context data within CPS (Requirement *R1*). Web services are the most important means of encapsulating and remotely invoking device functionality. Several works address the issue of dynamic service invocation due to unreliable and limited resources of IoT devices (Requirement *R2*). The aspect of ubiquitous human interaction in the context of business processes is only discussed in few works (Requirement *R3*). The importance of mobile and distributed process execution also increased with the development of more decentralized systems (Requirement *R4*). Most workflow systems proposed are proprietary developments resulting from the necessity of adding new software components to realize specific IoT or CPS functionality to WfMSes. We have also seen alternatives to BPMN and WS-BPEL for describing the underlying IoT-related processes. Despite being an important aspect in CPS, real-time is addressed by almost no related approaches in the workflow context. Flexibility in business process systems was identified as an important aspect by various works. As discussed in Section 2.4.6 this aspect also plays an important role in the smart home domain and in CPS in general due to the dynamic availability of resources and emergence of new situations. Ad-hoc changes to process instances as well as structural adaptations and evolution of process models [RWRW05] as proposed within the works related to the *ADEPT* project have to be considered complementary to our investigations, which focus on the CPS resources. The CPS-related requirements of synchronization, error handling, self-management and retrofitting (Requirements *R5–R8*) are not addressed explicitly by the investigated works.

3.5. Cyber-physical Synchronization

After evaluating related work regarding the design and implementation/configuration phases of the BPM lifecycle [VDA13], we move the investigations towards more runtime-oriented aspects. Under the term *Cyber-physical Synchronization*, we understand the issue of projecting a physical entity's state and properties to its virtual representation and vice versa as well as maintaining the consistency between both models in case of changes to one of the other (Requirements *R5* and *R6*). Our focus is on investigating *Cyber-physical Synchronization* in the context of workflows and (business) processes [Wom11a, Wom11b, MMS14, MDCM17] but we will also evaluate more general approaches regarding physical objects, (environmental) context factors and systems [PLM16, CR15, RvWLB15, CSB16, Sto15, RSI+17, dR12].

The aspect of *Cyber-physical Synchronization* is closely related to representing and synchronizing physical things, systems and environments to virtual entities– referred to as *Cyber-physical Objects* [PLM16], *Cyber-physical Equivalence* [Sto15], *Augmented Worlds* [CR15], *Thing/Device Shadows*[17], *Cyber Twins* [LBK15] and most prominently as *Digital Twins* [RvWLB15].

In [PLM16] Petrolo et al. describe *Cyber-physical Objects* as key elements and actors of CPS. Similar to *Smart Objects* [KKSF10], cyber-physical objects are aware of their surroundings, able to interact with users and to react to events. The authors

[17] `https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html`

discuss possible applications and programming models as well as basic platforms for implementing cyber-physical objects and CPS in general.

Stork discusses challenges regarding the realization of *Cyber-physical Equivalence* for Industry 4.0 applications in the context of visual computing [Sto15]. He argues that all characteristics of physical objects have to be represented by their virtual equivalences including geometry, functionality and behaviour. In case of a mismatch between the physical and digital objects, the corresponding models have to be updated or actions have to be performed to restore consistency between both entities. In addition, Lee et al. propose to use sensor data from production machines and their contexts to create a *Cyber Twin* for the machines of a smart factory [LBK15].

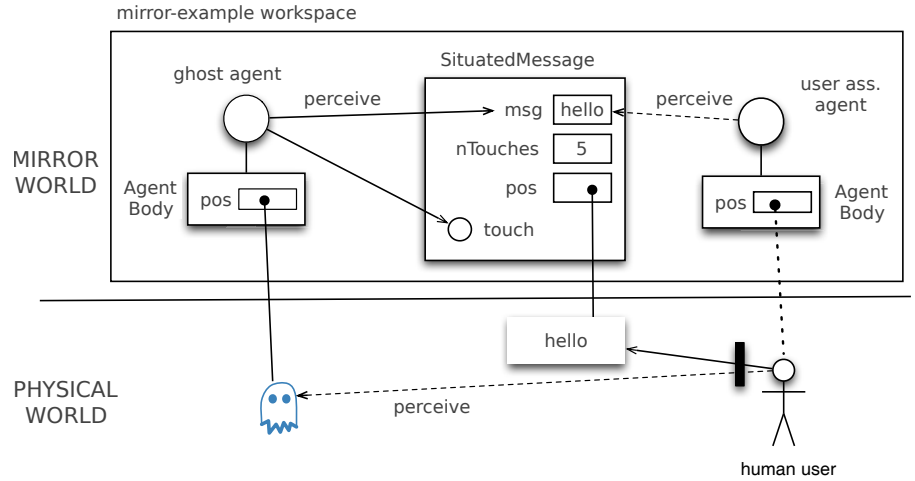


Figure 3.6.: Example for a Mirror World from [CR15].

Croatti and Ricci present their work regarding programming abstractions for augmented worlds in [CR15]. Their concept of *Augmented Worlds* links a computational object to its physical location. Such an augmented object–including its state and behaviour–can be represented and implemented through concepts of classical object-oriented programming approaches (i. e., classes and objects). Additional models have to be applied to handle concurrency, though. Mirror worlds [RPTC15] are the software-level representations of physical worlds and provide services to users based on software agents and the user's physical location. As shown in Figure 3.6, location-based (*situated*) messages are contained at specific points in the mirror world. Once the user reaches a physical location associated with a situated message, its mobile assistant user agent–running either on a smartphone or smart glasses–interacts with the corresponding agent in order for the user to perceive the message. There also exist *ghost agents* autonomously moving along the streets and interacting with the location-based services/agents or hugging the human users upon their encounter.

Among others, Rosen et al. discuss the importance of *Digital Twins* for future IT-based manufacturing solutions in the context of Industry 4.0 [RvWLB15]. Digital Twins are complete representations of physical products and systems including structured semantic information created throughout all stages of the lifecycle (*Design, Build, Operate*) of the respective system or product. This information is contained within a *Digital Thread* and available to all subsequent stages of the lifecycle. According to the authors, Digital Twins are key enabler for realizing and optimizing

Listing 3.2: Device Shadow Example for the State of a Traffic Light.

```
{    "state": {
        "desired": {
            "lights": {
                "color": "RED"  },
            "engine": "ON"  },
        "reported": {
            "lights": {
                "color": "GREEN"  },
            "engine": "ON"  },
        "delta": {
            "lights": {
                "color": "RED"  }
    }  },
    "metadata": {       ...
        "delta": {
            "lights": {
                "color": {
                    "timestamp": 123456
    }  }  }  },
    "version": 10,
    "timestamp": 123456789 }
```

Industry 4.0 solutions, e. g., production intelligence, production planning, product design and production system engineering [RvWLB15].

Practical solutions closely related to the idea of cyber-physical synchronization are already provided by various existing commercial IoT platforms to represent properties and states of things and devices. The Amazon AWS IoT platform[18] integrates the concept of *Thing Shadows* or *Device Shadows*, which are documents containing state information for IoT entities (things, devices, apps, etc.) that can be queried via web services. Listing 3.2 shows an excerpt of a Device Shadow for a traffic light[19]. It contains its states, metadata and also information about mismatches regarding the *desired* and the *reported* state. Similarly, the Eclipse Ditto project for development of IoT solutions features Digital Twins[20] to represent a real world device or asset with all its capabilities and aspects as virtual entity. Chang et al. propose to use specific *Virtual Thing Adaptors* to integrate physical (manufacturing) systems into digital software systems [CSB16].

As an extension to the *Physical Web* for IoT presented in [WSJ15] where people, places and things have webpages and services to interact with, Ruta et al. describe their vision of the Physical *Semantic* Web in [RSI+17]. Their framework relies on semantic descriptions being exposed and propagated by representations of things and humans to enable dynamic knowledge discovery and sharing in the IoT. Services and suitable entities can be found dynamically by logics-based queries with respect to specific capabilities and context constraints.

[18] https://aws.amazon.com/iot/?nc1=h_ls
[19] https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-document.html
[20] https://www.eclipse.org/ditto/intro-digitaltwins.html

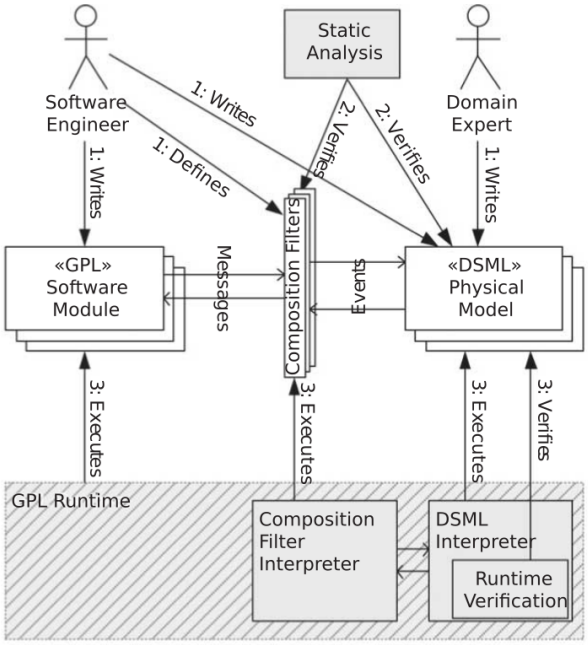


Figure 3.7.: Overview of the Verification and Analysis Process of Physical Models from [DRSA12].

De Roo et al. discuss in [DRSA12, dRSA11, dRSA14] various aspects of composing physical models with virtual models in embedded control software. They show how to formally verify the physical models at runtime with the help of sensor information and events, and adapt the models in case of inconsistencies for a cyber-physical example, namely the heating process of a laser printer. Figure 3.7 gives an overview of their approach. Software engineer and domain expert write the physical domain-specific models, whereas the general purpose software module is implemented by the software engineer. Composition filters are used for composing the physical models and software models. They communicate with the physical models via events and with the software modules via messages. Static analysis is used to verify the physical models and composition filters at design time and a special interpreter for the domain-specific physical models performs the runtime verification.

Wombacher discusses the correlation of physical objects and business processes in [Wom11b] and the issue of detecting potential errors in the corresponding sensor infrastructure in [Wom11a]. He distinguishes between two possibilities of correlating workflow and sensor data: a) to *trigger* a state transition in the control flow of a process based on sensor data; and b) to *monitor* the effects of the workflow instance execution based on changes within the corresponding sensor data. From that, Wombacher derives multiple classes of potential errors [Wom11b]:

- Successful workflow execution with reliable sensor data, i. e., everything is ok;
- successful workflow execution with unreliable/erroneous sensor data, i. e., the workflow was executed successfully but the sensor data associated with the workflow is incorrect;

- successful workflow execution with undocumented changes, i. e., ad-hoc changes within a specific workflow instance are not documented, but the instance is executed successfully;
- effect of workflow evolution, i. e., the workflow is adapted and applied to new resources that are not able generate the necessary sensor data;
- effect of changes on sensor infrastructure, i. e., faulty or broken sensors lead to corrupt data that cannot be correlated to the respective workflow instances.

In [MDCM17] Meroni et al. present an artefact-driven approach to monitor business processes. Artefacts of a BPMN process are enriched with additional data and information regarding their runtime states and process-related events to enable the monitoring and compliance checking of the artefact's and process instance's lifecycle among all involved process participants. They use a scenario from logistics as running example, which is implemented by an IoT software infrastructure.

As part of their *SmartPM* process management system [MMS14], Marrella et al. apply the Situation Calculus [LPR98] to formally specify preconditions and post-conditions of the execution of process activities. That way, they are able to determine deviations of the expected outcome of the process execution from its actual effects on context factors or process resources. A more detailed description of the *SmartPM* system is given in Section 3.6.

Conclusion

Table 3.4.: Evaluation of Related CPS Cyber-physical Synchronization Approaches for Workflows with respect to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[PLM16]	o	-	o	-	o	-	-	-
[Sto15]	-	-	-	-	+	-	-	-
[CR15]	-	+	o	-	+	-	-	-
[RvWLB15]	-	-	-	-	+	-	-	-
[RSI+17]	-	++	+	-	o	-	-	-
[DRSA12]	+	o	-	-	+	+	-	-
[Wom11b]	+	-	-	-	+	o	-	-
[MDCM17]	+	o	o	-	+	-	-	-

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

Table 3.4 presents an overview and evaluation of work related to the aspect of Cyber-physical Synchronization for WfMSes that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

From the investigations above, we see that only few approaches discuss aspects related to cyber-physical synchronization and the maintenance of *Cyber-physical Consistency* (cf. Section 4.6) in the context of workflows (Requirements *R5* and *R6*). With software-controlled CPS and IoT environments influencing the physical world, a need for representing physical entities, their states and contexts by digital models and instances (*Digital Twins*) was identified by several works. However, workflows and business processes are only abstract virtual concepts without a corresponding physical equivalent (*Physical Twin*). The concepts of subprocesses, control flow logic, events as well as pools and lanes can only partially be mapped to physical equivalences. Therefore, several approaches propose to use additional, possibly redundant sensor data from several sources to reliably correlate the effects of the execution of cyber-physical workflows to changes within the states of the physical process resources (things, humans and other physical entities including their contexts) and the physical environment. In case of inconsistencies between the expected outcome and the actual outcome of the process executions determined by additional sensor data, the information of the virtual representations have to be updated or compensation actions have to be executed in the physical world to restore *Cyber-physical Consistency* (cf. Chapter 6). Approaches for realizing this functionality autonomously are discussed in the following section. We will elaborate on the relation between CPS workflows and Cyber-physical Objects/Digital Twins in Section 6.10.

3.6. Self-* for BPM Systems

CPS and IoT environments are characterized as being highly dynamic and also vulnerable and prone to errors due to scarce and unreliable resources as well as unpredictable situations and behaviour of its constituents (cf. Section 2.4.6). Self-* capabilities including self-awareness, self-healing, self-optimization, and self-adaptation in general are therefore important properties required for autonomously acting systems and devices within CPS–either to restore inconsistent physical and cyber states (Requirement *R6*) or to handle errors and other unanticipated situations in general (Requirement *R7*). In this section, we discuss related work with respect to the aspect of *Self-** (Self-x) for adaptive (cyber-physical) software systems in general [BDK+15, BDK+17, RBD+09] and with a special focus on self-* capabilities for (cyber-physical) BPMSes [MMP06, OCEP13, Fri11, WSBL15, MMS14, RSA10, HSDV13, DTB+15]. These discussions mostly refer to the *Run & Adjust* phase of the BPM lifecycle as they deal with ad-hoc modifications to particular process instances and software configurations at runtime.

Complementary to the literature study presented in this section, a systematic literature review on self-adaptation for CPS by Muccini et al. can be found in [MSW16] and a discussion of challenges regarding self-adaptive CPS by Giese in [Gie16]. A survey on context-aware workflow adaptations by Smanchat et al. can be found in [SLI08] and a survey on self-healing systems by Ghosh et al. in [GSRU07] and by Psaier et al. in [PD11]. In [OG17] Oberhauser and Grambow present a comprehensive vision towards autonomically-capable processes supporting various self-* properties. The authors discuss potential technologies and methods to implement parts of the required components and functionality to realize this vision.

With *MUSIC*, Rouvoy et al. present a middleware for self-adaptation in ubiquitous and service-oriented environments [RBD+09]. The component-based architecture of the *MUSIC* platform integrates components for the management of context information and related Quality of Service (QoS) constraints, an adaptation manager using reasoning to find adaptation plans from a repository, and an executor to perform necessary reconfigurations, which results in an implementation of the MAPE-K feedback loop (cf. Section 2.4.5) based on the Deming cycle [Joh02]. The platform supports the dynamic discovery and binding of services, the negotiation and monitoring of Service-level Agreements (SLAs), and the hosting of services.

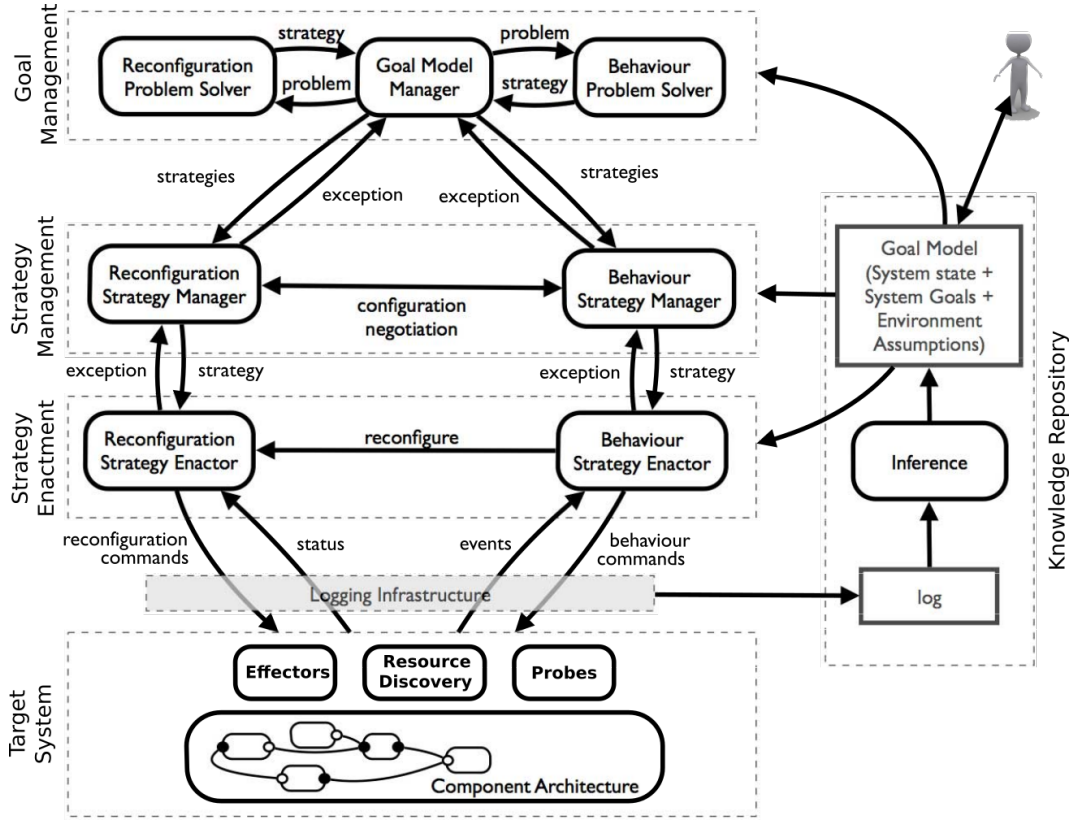


Figure 3.8.: The MORPH Reference Architecture from [BDK+15].

An additional general software framework for context-adaptive systems with self-adaptive behaviour is the *MORPH* reference architecture presented by Braberman et al. in [BDK+15, BDK+17]. Figure 3.8 shows the MORPH reference architecture, which is based on Kramer and Magee's three layer architecture for self-managed systems [KM07]. On the lowest level, the component-based target system emits events, status and sensor data, and other logging information (*Probes*), and it is interacted with via *Effectors* receiving reconfiguration and behaviour commands. These commands are deduced based on higher-level goals from the *Goal Management* layer defined by the users or inferred by the system. From these goals, strategies for dealing with exceptions are derived on the *Strategy Management* layer and enacted on the target system's effectors from within the *Strategy Enactment* layer.

With *SH-BPEL*, Modafferi et al. describe a self-healing plug-in for WS-BPEL engines [MMP06]. The system's process manager component integrates a management engine, which is able to retrieve and invoke substitute web services in case of errors. A *Diagnoser* component monitors the WS-BPEL engine and reports its diagnoses to a *Recovery Selector* component. Recovery strategies can be either directly implemented as part of the process manager or they can be triggered by external systems. Based on the diagnosis and selected recovery strategy, web services are chosen to be invoked by the workflow engine. With *Worklets* [ATHEVDA06] and *Exlets* [ATHVDAE07] Adams et al. present SOA-based approaches to dynamically select subprocesses according to the context of the current workflow tasks and exceptions occurred during the process executions from special repositories.

D-OSyRIS by Frîncu is a self-healing distributed workflow engine [Fri11] for pervasive and mobile computing environments. The workflow engine relies on the *SiLK* language to model workflows, which is a nature-inspired rule-based language describing workflows by means of chemical reaction-like behaviour [FP10]. The author presents a decentralized workflow system, which also features a healing mechanism based on the Deming cycle to find unresponsive workflow engines and replace them. The *Monitor* gathers context information from all involved workflow engines; the *Analyser* determines failed modules; the *Planner* decides on which resource a failed component can be restarted; and the *Executor* restarts the failed component.

Chang and Ling present a similar context-aware solution to replace failed devices in service-oriented workflows [CL08] implemented in the *Decoflow* engine [Lok03]. The dynamic replacement of faulty services and devices in decentralized IoT-aware business processes is proposed by Dar et al. in [DTB+15]. An approach for migrating business processes among peers of a distributed industrial IoT device landscape in case of device failures or disconnections is discussed by Mass et al. in [MCS16].

Hoenisch et al. present their work on self-adaptive resource allocation for elastic process execution in [HSDV13]. The goal is to automatically adapt required Cloud resources for running the underlying BPMS (*ViePEP* [SHVD12]) based on the current demand and current and future process landscape. The authors propose to implement the MAPE-K feedback loop to enable self-healing, self-configuration and self-optimization. In this particular work, resource utilization of virtual machines running the process execution system is monitored and analysed with respect to specific SLAs. The planner takes the current load but also future workflow steps and their associated predicted loads into account to derive a scheduling and resource allocation plan via reasoning.

The *MABUP* approach for multi-level autonomic business process management is discussed by Oliveira et al. in [OCEP13]. The *MABUP* system comprises four levels of modelling a business process: the organisational level, the technological level, the operational level and the service level. During process execution, a dedicated MAPE-K-based management phase supervises and instruments the execution system. The *Monitor* collects sensor information regarding context, domain-specific and non-functional requirements as well as QoS criteria. The *Analyser* determines deviations from expected behaviour as defined within the business process's domain assumptions and context criteria. The *Planner* selects operational tasks to be executed to compensate the deviations occurred for the respective instances, and the actuator components enact these compensations as part of the *Executor*.

Richly et al. present in [RSA10] a semantic BDI (Belief-Desire-Intention)-based approach [RG+95] to realize cooperative and reflexive workflows. The approach uses the BDI technique in combination with semantics to implement autonomous workflow adaptations for the OSPP system–a service-based workflow engine with WS-BPEL-like processes [HRR+08]. In this work, the *Belief* includes the context variables and the executed workflow. *Intentions* are represented by available workflows and *Desires* are internal objects of the system. With these information, the workflow system is able to learn new workflow cases and adaptations at runtime as well as merge workflow instances to reduce redundant task execution.

In [Sch09] Schlegel describes an approach for object-oriented interactive processes in decentralized production systems. The work is based on a semantic object-oriented process metamodel called *OMICRON* [Sch08] to create variants of activities by inheritance and extensions in case dynamic changes to the respective process are required. The system also supports the distribution of processes and models as well as the generation of user interfaces to interact with the processes. A more detailed discussion of business process adaptation mechanisms on the instance and type level can be found in [DR09, RRKD05].

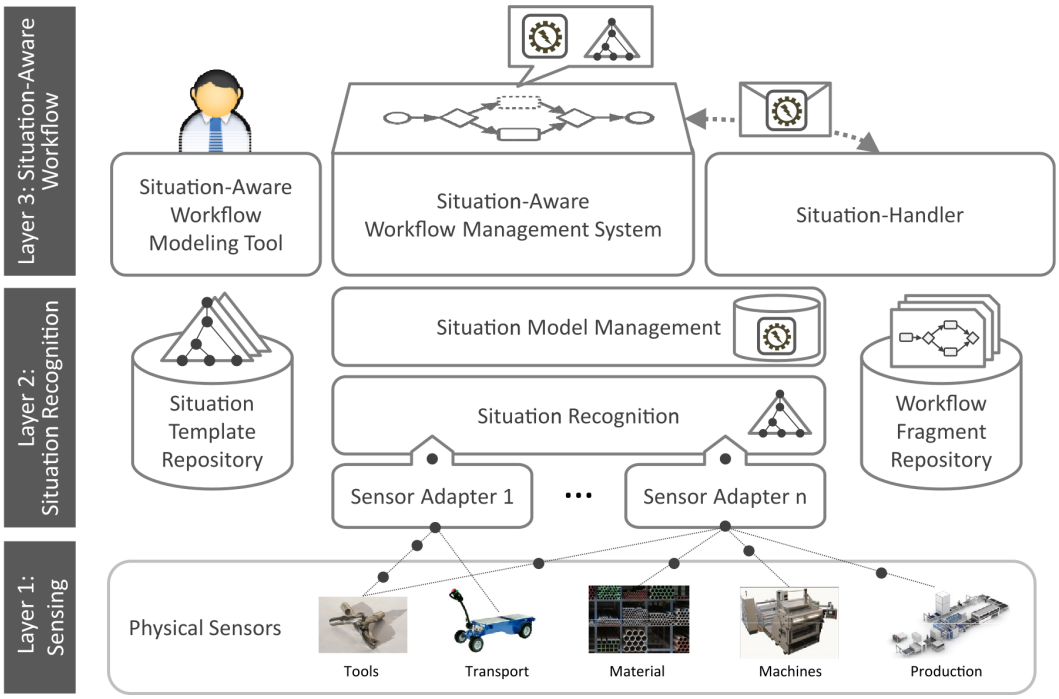


Figure 3.9.: Architecture of the SitOPT System and its Layers from [WSBL15].

With *SitOPT*, Wieland et al. developed a system for situation-aware adaptive workflows in production systems and CPS/IoT environments in general [WSBL15, HWS+16]. Figure 3.9 gives an overview of the system architecture of SitOPT. On the lowest layer (*Sensing*), sensors measure data from various physical entities of the CPS. Situations are recognized based on predefined descriptions and templates of possible situations after evaluating the sensor data and finding matching situations within the self-optimizing *Situation Recognition* layer. On the *Situation-aware*

Workflow layer, processes are modelled and executed by the respective WfMS. In case of exceptions (e. g., machine failures) and other unintended situations that occurred during the process execution, suitable workflows or workflow fragments to handle the particular situation are selected from a workflow fragment repository that contains all possible workflows and remedies to specific situations.

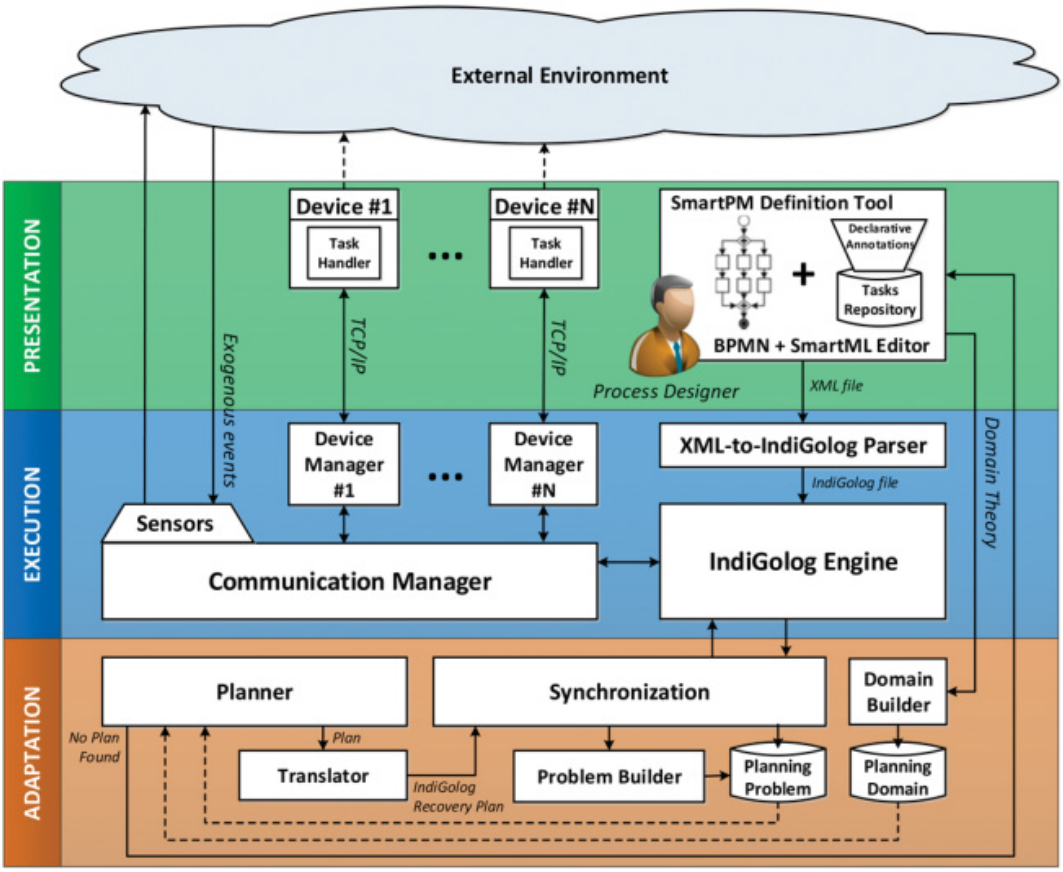


Figure 3.10.: The Architecture of SmartPM from [MMS16].

The *SmartPM* system developed by Marella et al. is a cognitive business process management platform for adaptive CPS processes [MMS14, MMS16, MMHS15, MMS17, MM17]. The authors approach the issue of adaptive and self-managed processes from the perspective of artificial intelligence for *Knowledge-intensive Processes* [DCMR12]. They use reasoning about actions, high-level programming and automated planning to implement automated adaptations of processes in highly dynamic settings (e. g., cyber-physical domains). Figure 3.10 shows the high-level architecture of the SmartPM system as an advancement of the *ASTRO-CAptEvo* system enabling dynamic context-aware adaptation for service-based systems [RBK+12]. Devices interact with the external environment and BPMN-based workflows for SmartPM are defined on the *Presentation* layer. The workflows are described based on the *CAPTLang* language for context-aware adaptable business processes [BMP13] and translated into the IndiGolog programming language for embedded reasoning agents [DGLLS09]. They are executed by the respective *IndiGolog Engine*, which then communicates with the respective devices via the *Communication Manager*

component. Context factors, preconditions and postconditions of workflow activities as well as situations before and after the execution of these activities are formalized using the Situation Calculus [LPR98]. At runtime, workflow-related events from sensors are analysed with respect to these situations. In case a mismatch between the physical reality and expected reality can be determined, the *Planner* component is triggered to find a recovery procedure consisting of new workflow tasks. The recovery plans and adaptations are determined by using *Classical Planning* to solve the planning problem of getting from the current state to the desired state [Mar17].

Conclusion

Table 3.5.: Evaluation of Related Self-* Approaches for Workflows with respect to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[RBD+09]	+	+	-	-	-	-	++	-
[BDK+15]	o	o	-	-	o	o	++	-
[MMP06]	-	+	-	-	o	o	+	-
[Fri11]	-	o	-	++	-	-	+	-
[CL08]	-	+	-	-	-	-	o	-
[HSDV13]	o	+	-	o	-	-	++	-
[OCEP13]	o	o	-	-	-	-	+	-
[RSA10]	o	-	-	-	-	-	++	-
[Sch09]	-	-	+	+	-	-	o	-
[WSBL15]	+	+	o	-	o	o	+	-
[MMS16]	+	+	o	-	++	+	+	-

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

Table 3.5 presents an overview and evaluation of work related to implementing self-* properties for WfMSes that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

From the investigations, we found several approaches dealing with aspects related to the realization of self-* properties for software systems in general and for WfMSes in particular–also in the context of CPS and IoT (Requirement *R7*). The topic of self-adaptive and self-managed software systems in general is a very broad and diverse topic with general concepts and more concrete approaches regarding specific domains and problems. The implementation of the Deming cycle in the form of the MAPE-K feedback loop is a widely accepted approach to achieve certain degrees of self-adaptation and other self-management capabilities for software and workflow systems in specific application domains. Domain and context models describe the environment the WfMS operates in. Goals and other formalisms are used to spec-

ify the expected outcome of the workflow executions. By analysis of goal-related data from additional external and internal sensors, mismatches, (cyber-physical) inconsistencies and unexpected situations can be determined (Requirement *R5*) and compensation strategies can be derived by planning components to remedy these issues (Requirements *R6* and *R7*). Several approaches rely on adapting the process by finding alternative resources or services to handle errors with respect to the process resources. The selection or synthesis of new workflow fragments is another approach to adapt the respective process instances in case of undesired behaviour. While several approaches address self-healing and self-adaptation for software systems, workflows and IoT applications in case of device errors and inconsistencies, the aspects of human interaction and also distributed process execution are rarely discussed. The broad spectrum of approaches targeting the realization of self-* capabilities for software and workflow systems–ranging from the MAPE-K loop, object-oriented models, semantics-based adaptations to complex planning algorithms using artificial intelligence–shows the complexity of this particular aspect and research field with approaches targeting more general concepts but also concrete domain and application specific solutions.

3.7. Retrofitting Frameworks for WfMSes

In Section 3.2, we evaluated several existing WfMSes used in industry and academia with respect to the requirements. We found that there is a large number of WfMSes with no or very limited support of CPS-related features. As it is infeasible to completely redesign and modify these BPMSes or to replace them by other systems in order to use them in the context of CPS and IoT, we investigate at this point if there are solutions and approaches to retrofit these existing WfMSes and software systems with respect to the required CPS-related capabilities (Requirement *R8*).

In [Mon13] Monnier talks about smart grids enabled by the IoT. He briefly discusses the aspect of retrofitting existing meter infrastructures in houses with additional sensors and appliances to realise smart meters and smart grids. Also in the context of smart homes, Alur et al. mention the challenge of retrofitting existing houses with IoT devices [ABD+16]. When discussing the vision of smart cities, Harmon et al. [HCLB15] briefly mention the aspect of retrofitting the existing infrastructure with additional smart sensors and actuators [HCLB15]. In [AMT+12] Aswani et al. present a concrete example for retrofitting an air conditioner with a cyber-physical control system to improve energy efficiency. Camps et al. discuss the issues of adding autonomic agents based on control loops to increase the fault tolerance of health monitoring systems in [SH05]. Moctezuma et al. show in [MJPL12] how to retrofit a factory automation system to address new market needs and societal changes. They add new hardware components for monitoring and new software components based on web services to the existing production infrastructure in order to increase the self-awareness of the overall system and with that, to increase energy efficiency, reconfigurability, safety and other quality-related parameters.

A more general approach of retrofitting a legacy SCADA (Supervisory Control and Data Acquisition [DS99]) system with an external event-based coordination layer to add fault tolerance and protection against cyber attacks in cyber-physical environments is presented by Xiao et al. in [XRK08]. Kaiser et al. describe an

external infrastructure for monitoring distributed legacy systems and adding autonomic capabilities to these systems in [KPGV03]. Their approach relies on events to communicate asynchronously with probes and effectors of the legacy system. In a follow-up work, Parekh et al. developed a general methodology for retrofitting autonomic capabilities onto legacy systems [PKGV06]. Figure 3.11 gives an overview of their proposed retrofitting reference architecture. Sensors gather information (*Probes*) from the legacy systems that are collected and then interpreted and analysed by *Gauges*. Decision and coordination of possible adaptation strategies based on the results of the interpretation are conducted by the controllers, which instruct the effectors of the legacy system in case reconfigurations are required.

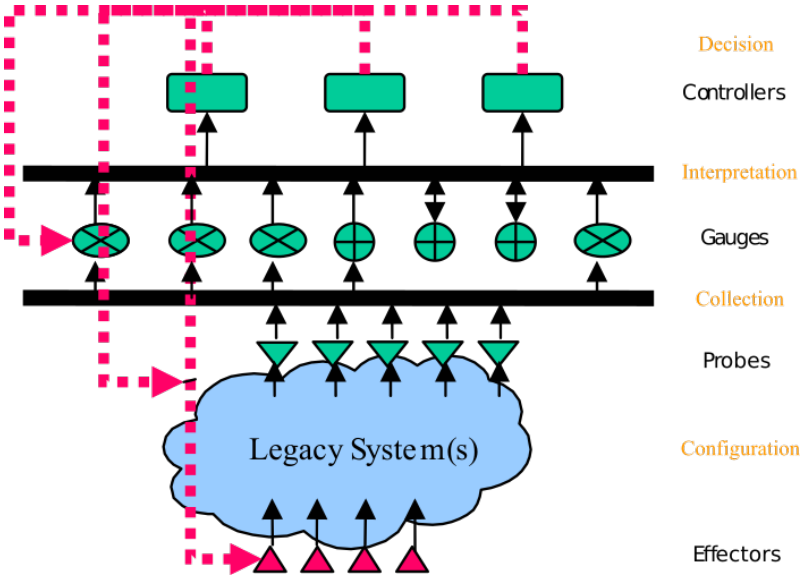


Figure 3.11.: Reference Architecture of the Retrofitted Autonomic Software Infrastructure from [KPGV03].

Dabholkar and Gokhale present in [DG09] an approach of specializing general purpose middleware systems with respect to specific requirements that CPS demand of a software system (e. g., real-time behaviour and dealing with constraint resources). They use feature-oriented software development to abstract required features and generative programming to realize the specialized middleware. A general architecture for the implementation of CPS that can also serve as a framework for retrofitting legacy systems is discussed by Lee et al. in [LBK15] and described in Section 2.4.3.

In [BtHS01] Barros et al. show how to retrofit internal assembly processes with more high-level business processes based on components and services to facilitate synchronization across encapsulated workflows and external interactions. Lee et al. present a way to add adaptivity to an existing scientific workflow engine by implementing new components to realize the MAPE-K feedback loop [LPS+09]. Sensors provide data about the status of jobs running on the computation grid. Once resources are available, the workflow engine is instructed to execute new workflows.

Conclusion

Table 3.6 presents an overview and evaluation of work related to retrofitting of WfMSes with self-* capabilities that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-).

The investigations of related work with respect to the aspect of retrofitting existing WfMSes and software systems in general with CPS-related capabilities (Requirement *R8*) have shown only few relevant approaches. Most works present concrete ways to retrofit existing hardware and software infrastructures with new devices and software components in the context of CPS to make them more self-aware and efficient. The most common approach towards retrofitting is to loosely add new components that implement a variant of the MAPE-K feedback loop, which serves as a general framework to facilitate autonomous behaviour in WfMSes and software systems. However, there is no concrete suggestion regarding the retrofitting of existing BPMSes with new capabilities to implement autonomous (CPS) workflows.

Table 3.6.: Evaluation of Related Retrofitting Approaches w. r. t. Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[XRK08]	o	-	-	-	-	-	-	o
[PKGV06]	+	o	-	-	-	-	+	+
[LBK15]	o	o	-	-	-	-	+	o
[LPS+09]	o	o	-	-	-	-	+	+

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

3.8. Conclusion & Deficits

Table 3.7 presents a summarizing overview and evaluation of all related works that we investigated with respect to the identified requirements. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-). From Tables 3.1 and 3.7 we see that many existing systems and related works were developed that address relevant aspects to realize workflows for CPS and IoT. Most of these approaches sufficiently cover single or multiple requirements identified in Section 2.6 across the whole BPM lifecycle. However, there is no comprehensive solution that addresses all requirements in the context of BPM that are relevant for this thesis.

Nevertheless, we are able to deduce important concepts, methods, technologies, components and frameworks suggested as suitable solutions by the investigated works to design, implement and operate a suitable WfMS for CPS and IoT covering all of the relevant requirements:

- In order to monitor and evaluate simple and complex streams of sensor data from CPS entities, simple event processing (ECA rules) and CEP can be applied. (**Requirement *R1***)
- In order to dynamically find and invoke process resources in the CPS, web services in combination with a service registry and semantic description of the services and CPS entities can be applied. (**Requirement *R2***)
- In order to enable ubiquitous interaction with users, synchronous and asynchronous communication as well as advanced mobile and stationary user interface technologies can be applied. (**Requirement *R3***)
- In order to realize distributed process execution, decentralized systems and coordination infrastructures in combination with process instance migration can be applied. (**Requirement *R4***)
- In order to synchronize the physical and virtual worlds, additional sensor information in combination with formalisms to describe effects of workflow tasks and CPS entities can be applied. (**Requirement *R5***)
- In order to deal with errors and other unanticipated situations during CPS process execution, the MAPE-K feedback loop from autonomous computing can be applied. (**Requirement *R6***)
- In order to add more general self-* capabilities to the CPS workflow system, the MAPE-K feedback loop in combination with goal specifications can be applied. (**Requirement *R7***)
- In order to retrofit existing WfMSes with CPS-related capabilities, loose service-based coupling of the legacy system with the MAPE-K feedback loop and external event and data sources can be applied. (**Requirement *R8***)

All of these individual approaches and components require new concepts and a holistic engineering approach to combine them with respect to the relevant phases of the BPM lifecycle, which are presented in the following chapters:

- The modelling of CPS workflows and related CPS entities (cf. Chapter 4);
- The implementation of a CPS worfklow management system and related IoT infrastructure (cf. Chapter 5);
- The operation of the WfMS and the autonomous execution of the workflow instances (cf. Chapter 6).

In these chapters, we will discuss the application of the aforementioned concepts for realizing the individual requirements by a comprehensive WfMS in detail. We will elaborate on the respective design decisions and the implementation of the individual software components to realize the desired functionality as well as alternative solutions. As already discussed in Section 2.6, the findings from related work are usually deduced based on application domains other than the smart home. However, the concepts are rather general approaches that can be easily transferred to other CPS domains, especially the smart home as it shows very similar characteristics as other smart spaces [KBG13].

Table 3.7.: Evaluation of Related Work with respect to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[DMC14]	+	o	o	-	-	-	-	-
[KSKP11]	+	-	o	-	-	-	-	-
[SSOK13]	+	+	o	o	-	-	-	-
[BBDC+15]	++	o	o	-	-	-	-	-
[AKF+14]	++	o	o	-	-	-	-	-
[MRM13]	+	+	o	-	-	-	-	-
[YBSD16]	+	o	++	-	-	-	-	-
[GKGK16]	o	+	o	-	-	-	-	-
[BDGP17]	+	o	o	-	-	-	-	-
[MD17]	o	o	o	-	-	-	-	-
[SGCG17]	o	+	+	-	-	-	-	-
[GEPF11]	o	o	-	-	-	-	-	-
[CS11]	+	o	-	-	-	-	-	-
[JDK15]	o	-	o	-	-	-	-	-
[MMG08]	-	++	-	++	-	-	-	-
[DTB+15]	+	+	o	+	-	-	+	-
[PRBA15]	+	o	+	+	-	-	-	-
[GCFP10]	+	o	++	o	-	-	-	-
[PRS+13]	+	+	o	+	-	-	-	-
[MCS16]	+	+	o	++	-	-	-	-
[PLM16]	o	-	o	-	o	-	-	-
[Sto15]	-	-	-	-	+	-	-	-
[CR15]	-	+	o	-	+	-	-	-
[RvWLB15]	-	-	-	-	+	-	-	-
[RSI+17]	-	++	+	-	o	-	-	-
[DRSA12]	+	o	-	-	+	+	-	-
[Wom11b]	+	-	-	-	+	o	-	-
[MDCM17]	+	o	o	-	+	-	-	-
[RBD+09]	+	+	-	-	-	-	++	-
[BDK+15]	o	o	-	-	o	o	++	-
[MMP06]	-	+	-	-	o	o	+	-
[Fri11]	-	o	-	++	-	-	+	-
[CL08]	-	+	-	-	-	-	o	-
[HSDV13]	o	+	-	o	-	-	++	-
[OCEP13]	o	o	-	-	-	-	+	-
[RSA10]	o	-	-	-	-	-	++	-
[Sch09]	-	-	+	+	-	-	o	-
[WSBL15]	+	+	o	-	o	o	+	-
[MMS16]	+	+	o	-	++	+	+	-
[XRK08]	o	-	-	-	-	-	-	o
[PKGV06]	+	o	-	-	-	-	+	+
[LBK15]	o	o	-	-	-	-	+	o
[LPS+09]	o	o	-	-	-	-	+	+

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

4. Modelling of Cyber-physical Workflows with Consistency Style Sheets

> "Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."
>
> Edsger W. Dijkstra

4.1. Introduction

This chapter focuses on various aspects regarding the modelling of workflows for cyber-physical systems. With the main goal being the automation of processes and activities in CPS involving sensors, actuators, complex machines, smart objects and humans, a more technical and thereby machine-processable imperative description of operative workflows is required. As pointed out in Chapter 2 and Section 3.3, existing business process languages (e. g., BPMN 2.0 or WS-BPEL) lack expressiveness with respect to aspects related to CPS and IoT, are too ambiguous to be interpreted in a standard way, and contain many elements that are not needed for CPS. The primary focus of these languages is on modelling *B*usiness *P*rocesses, which is why additional formalisms are required to model *Cyber-physical Processes*. In the following, we present the workflow metamodel used as the underlying formalism to describe workflows within the scope of this thesis. This workflow language is designed with a focus on the implementation perspective of CPS workflows [PDB+08], while still providing an abstract view on the underlying "business" processes describing the behaviour of the CPS. These workflows can be seen as templates or recipes that define the basic structure–including the flow of activations and data among the entities of CPS–of processes at design time. However, due to the very dynamic nature of CPS, instances of these processes have to be (self-)adaptive at runtime to deal with unanticipated behaviour, exceptions and errors (cf. Chapter 6).

The workflow language presented in this chapter supports the modelling of hierarchical workflows and detailed control and data flows. Special process steps are used to model the interaction with actuators via a variety of services and with multiple sensors via CEP. User tasks within processes and distributed processes can also be modelled with the help of the metamodel. We introduce a formalism to specify the expected (cyber-physical) effects of the execution of process steps and define criteria for potential failures. In addition, we present a semantic structure for modelling the CPS and IoT entities, their relations and context, which can be used to specify necessary requirements a resource has to fulfil in order to execute specific tasks.

4.2. Workflow Metamodel

With designing and developing a workflow management system for CPS, we rely on various principles from software engineering and model-driven software development. The metamodel follows principles from object-oriented programming and component-based software engineering to enable extensibility and reusability of workflow components. The basic structure of the workflow metamodel is described in detail in [SKNS13, SKNS15, SHS16].

4.2.1. Process Meta-Level Hierarchy

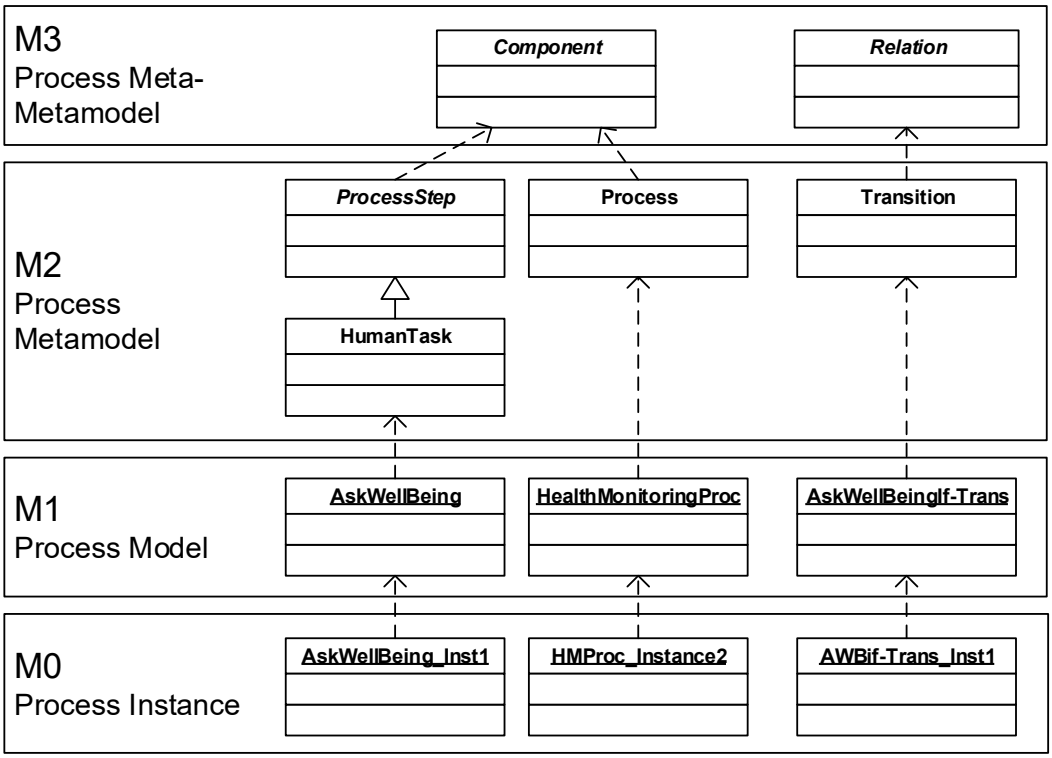


Figure 4.1.: Process Meta-Level Hierarchy.

Before describing the concepts and elements applied in the workflow metamodel in detail, we provide a classification of the different meta-levels for processes used in this work following the Meta-Object Facility (MOF) specifications [Omg08]. Figure 4.1 shows the runtime view on this hierarchy in UML notation with exemplary elements on each level. The relationships between two consecutive meta-levels are *instanceOf*-relations from the lower to the upper level. Based on the findings from the *OMICRON* meta-metamodel for object-oriented processes described in [Sch08], we assume that *Components* and *Relations* are sufficient meta-metamodel elements (Level *M3*) to be used for defining the basis for our workflow metamodel (Level *M2*). These *M3* level classes are instantiated on level *M2* to describe the process metamodel elements, which will be described in detail in the following sections. These classes are used to create (multiple) models of processes (Level *M1*) describing

an abstract workflow (e. g., the *Emergency Process* in the smart home from Section 2.2.2). From these process models, the WfMS creates (multiple) process instances (Level M0) for each process model during process execution at runtime. Levels *M3* to *M1* are related to the design time view, whereas level *M0* shows the runtime view on processes. The elements in the individual meta-levels represent only an excerpt from the complete set of elements, which will be described in the following with a focus on process elements on the metamodel level *M2*. In this chapter, we will describe core metaclasses regarding the structure of processes and special process steps that are relevant for discussing the requirements *R1–R8*. The complete process metamodel and additional explanations of important classes–however not of the entire metamodel–can be found in Section C.

4.2.2. Composite Structure

The basic building block and central element of the metamodel (Level *M2*) is the *Process Step*. As workflows in CPS can become very complex, a composition of process steps is necessary to enable the organizational and functional structuring of workflows. Following the *Composite* design pattern [Gam95], we distinguish between abstract *Composite* steps containing one or more process steps (substeps) and *Atomic* steps as specializations of the abstract Process Step class (cf. Figure 4.2). Atomic steps (*Activities*) provide a black-box view on process activities, which cannot be decomposed any further. An atomic step usually encapsulates a single task, a function call, a call to a specific service method or micro-service [VGC+15].

Composite steps can be composed of other process steps–atomic steps or other composite steps (e. g., *Loops*). A *Process* is regarded as a composite step forming a closed, self-contained workflow including other subprocesses, composite steps and atomic steps. This modular structure leverages extensibility and reusability when modelling complex processes. However, the metamodel does not impose restrictions on the level of granularity for a process, a composite step or an atomic activity. The process designer is responsible for deciding about the process structure, granularity and composition of process steps based on available interfaces to interact with CPS devices and services.

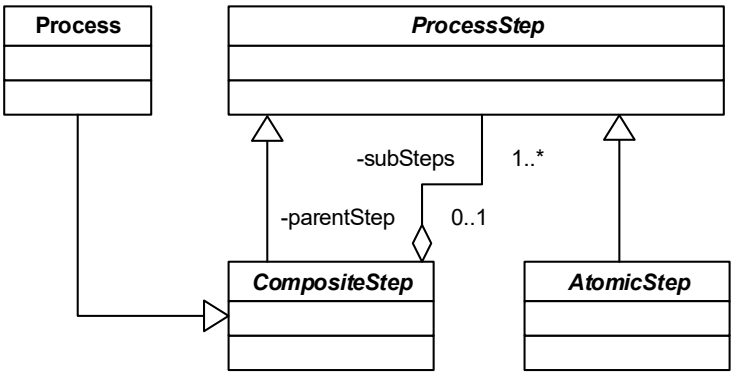


Figure 4.2.: Composite Components of the Core Process Metamodel.

4.2.3. Component Relations

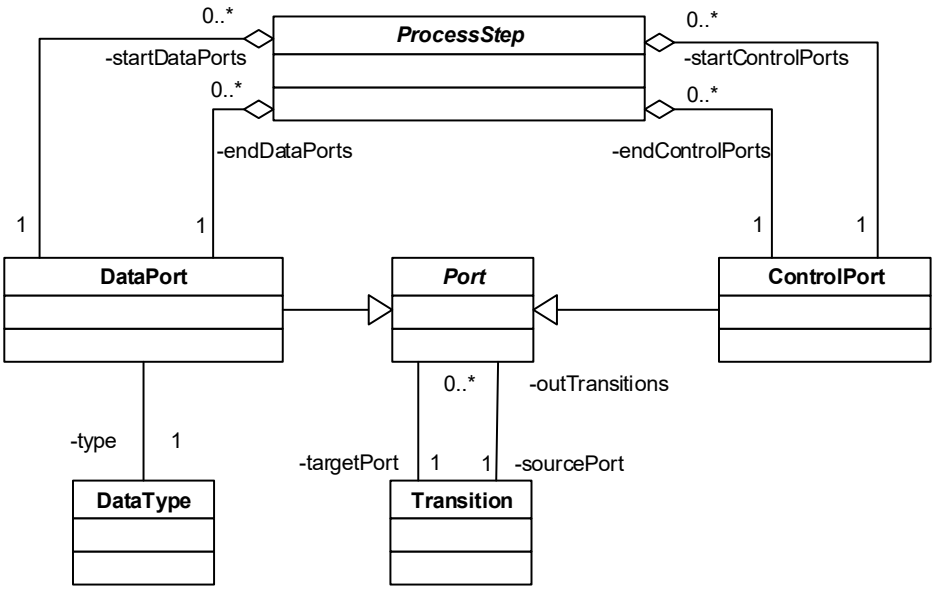


Figure 4.3.: Ports and Transitions between Process Steps.

Directed *Transitions* are used to model relations within a process as a directed graph defining the flow of activations and data along the edges between processes steps (nodes). In analogy to the concept of software components discussed by Szyperski et al. [SBW99], we view process steps as components providing interfaces–*Ports*– for ingoing and outgoing data (*Data Ports*) as well as for activations (*Control Ports*). A port represents an entry or exit point for data or control flow concerning a process step. This concept is adapted from the *Port* concept used in the WS-BPEL workflow language [OAS14] to define abstract interfaces for processes. By connecting these start and end ports of process steps with the help of *Transition* relations, the directed flow of activations between control ports and the flow of data between typed data ports–and therefore between process steps–can be defined (cf. Figure 4.3).

In general, we distinguish between Data Ports and Control Ports, which are both specializations of the *Port* class. Data ports are used for modelling data being consumed by process steps as input parameters or being produced by process steps as output values. Data ports represent data of a certain *Data Type*, which can also be modelled using the process metamodel to have an abstract process-level view on the data flow. To support the use of data objects of different types, multiple ingoing ports (*startDataPorts*) and outgoing ports (*endDataPorts*) can be contained within a process step. Control ports are used for connecting process steps that do not require the processing of ingoing or outgoing data. Diverging control or data flow can be modelled by using multiple outgoing ports. A process step can also contain multiple ingoing ports, which may be connected to multiple preceding process steps to merge control or data flow.

As shown in Figure 4.3, connections between process steps are modelled by using *Transitions*, which can be viewed as a directed relation between exactly one port of a process step (*sourcePort* association) and exactly one port of another process step

(*targetPort* association). A port contains references to all transitions that originate from it. When modelling composite process steps, there also needs to be a transition created between the start ports of parent step and its child step, as well as between their respective end ports. Transitions are only allowed to be created between the ports of distinct process steps and between process steps on the same hierarchical level and their direct parents. This cannot be enforced structurally by the model, but it has to be expressed by using additional constraints and constraint languages. Figure 4.4 shows an example of a graphical process model including process steps, which again contain other encapsulated processes that are connected via transitions.

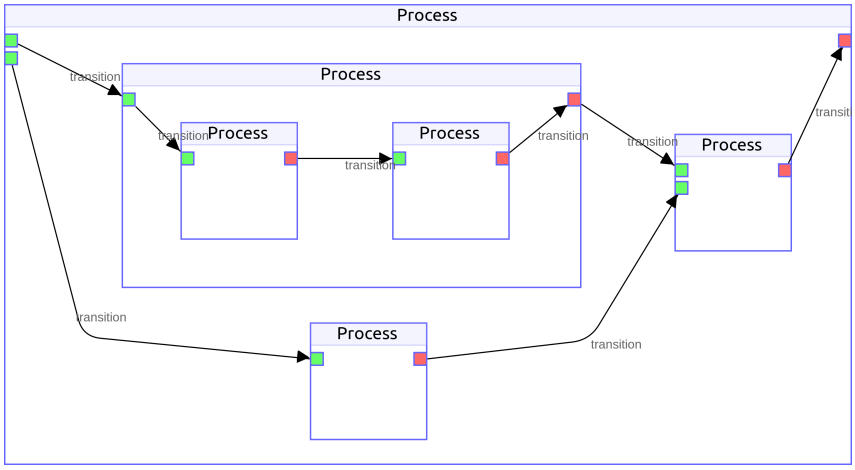


Figure 4.4.: Example of a Process Containing Several Connected Subprocesses.

4.2.4. Process Step Specializations

Thus far, the basic structure of processes has been described with focus on process steps as components and their composition. In order to increase the expressiveness of the workflow language, new elements have to be added to describe more complex control and data flow structures. Figure 4.5 depicts a small set of basic constructs from common programming languages as extensions that have been introduced as specializations of composite and atomic process steps. In general, a process step will be executed if all of its start ports are in an activated state, which corresponds to the logical *AND* operation. Other logical connectors for joining the control flow and modelling more complex activation patterns (e. g., *OR* and *XOR* connections) need to be modelled explicitly. Using the *If* directive, we can define conditional join operations based on data at the start ports of the respective process step and the subsequent activations of *TrueTransitions* and *FalseTransitions* w. r. t. the evaluation of the If-condition. The forking of control flow into parallel sequences is modelled by creating multiple outgoing ports and transitions from the corresponding process step to the target process steps (cf. Figure 4.4). *Loops* are viewed as extensions of composite process steps, which may contain arbitrary process steps. Extensions of this class are used to implement *Do–While*, *For* and *While* loops using loop conditions and counters to define loop behaviour and conditions for loop exits.

The object-oriented design of the metamodel facilitates the extensibility of the set of modelling elements. New specialized process steps can be easily introduced by extending the atomic or composite process step with new classes in an inheritance relation. Figure 4.5 shows extensions of the atomic process step to add service invocations (cf. Section 4.2.8) and data manipulations (cf. Section 4.2.6) to the set of supported process steps.

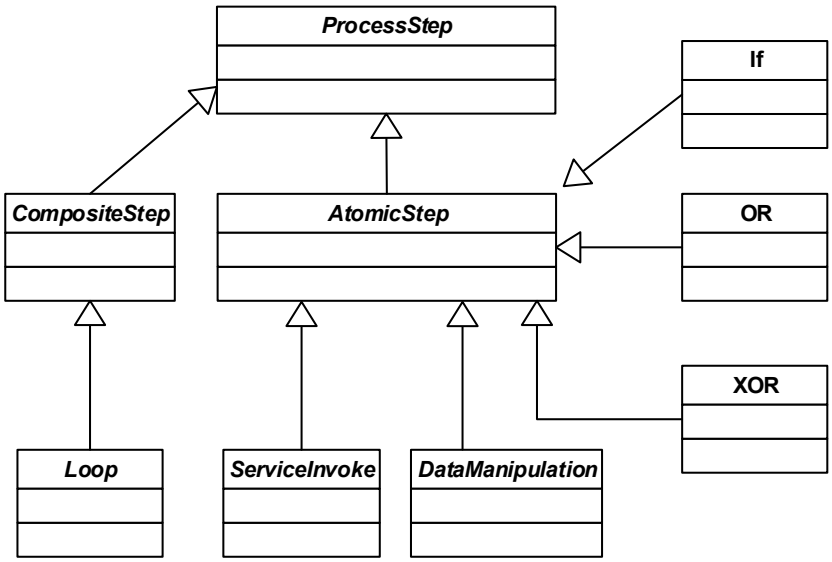


Figure 4.5.: Extensions of Process Steps via Inheritance.

4.2.5. Process Step Attributes

Process steps have additional attributes to further describe their properties. *Names* and *Unique identifiers* allow for managing process steps and instances. A text *Description* can be used to provide a description of the particular process step.

An optional *Resource* attribute specifies the resource responsible for executing the process step–in analogy to the *Swimlane/Pool* concepts from BPMN 2.0. This attribute is used primarily to provide a network identifier (IP address or host name) and it can be combined with an additional *Distributed* flag in case a specific process has to be executed on a remotely connected process engine (Requirement *R4*: Distributed Processes). To increase the flexibility of this rather static assignment, a type-based, capability-based or even role-based specification of a resource or its properties, respectively, could facilitate the dynamic selection of process resources at runtime (Requirement *R2*: Dynamic Resources) [HSKS16b, Sch08].

By adding a domain specific *Type* attribute to the metaclass, the semantics of a *Process Step* can be further described. A corresponding *Process Ontology* has to be developed to classify types of process steps (e. g., stating that the current process step is a *Cleaning* process or a *Fetching* process in the Smart Home domain) and relations among process steps. This classification can then be used to enable runtime adaptation and composition of processes (e. g. based on inheritance or aggregation of process steps) as described in [Sch08, SKNS15].

4.2.6. Data Types and Data Flow

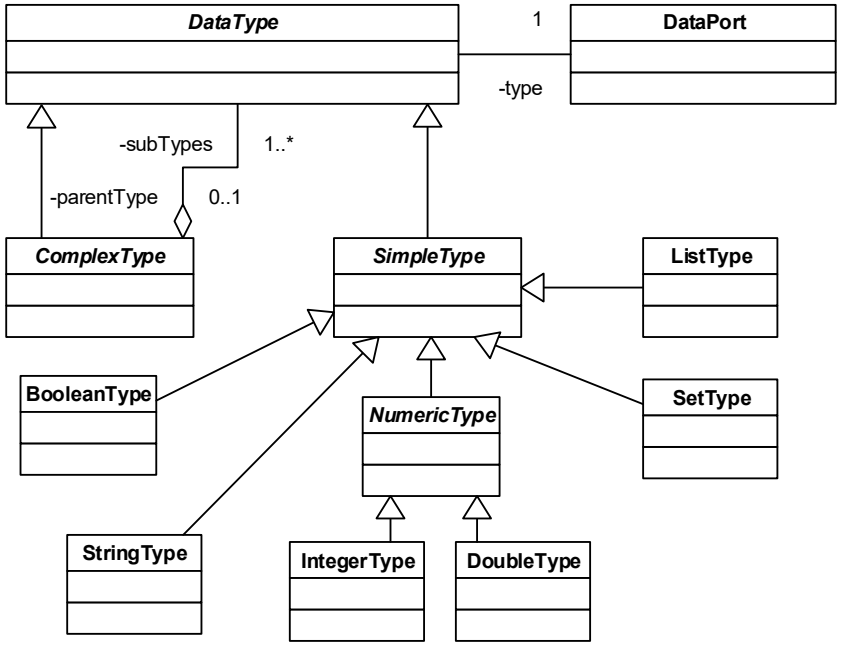


Figure 4.6.: Metamodel for Process Data Definition.

Automating the processing and routing of data flow among process steps, services, sensors and actuators as well as humans requires more formal descriptions of data flow and input/output of process steps than it is possible to model with state-of-the-art workflow languages (e. g., BPMN 2.0 or WS-BPEL). To provide a uniform view on the data flow, the process metamodel also includes means for defining simple data types and mappings of input and output data of process steps. In analogy to the composite pattern used for process components, these data types comprise simple atomic types (e. g., Integer, Boolean, String, etc.) and more complex self-defined types including lists, sets and combinations of these simple types (cf. Figure 4.6) in XML and JavaScript Object Notation (JSON)-like tree structures. With this, we model the data flow between ports on the process level, which is transformed automatically to input/output parameters for the specific service calls corresponding to the process step subclass or presented as input/output forms to the user in case of human activities. When connecting two data ports with the help of a transition, type compatibility of these ports is automatically checked by the process modelling environment (cf. Section 4.8.1). Data type definitions are contained in a *Process* and linked to the particular data port by means of an association. A basic form of scoping for data types and data instances can be achieved by creating a data type definition in a particular (sub)process, which restricts access to this data type to data ports of the same process and its child processes, but not to its parent processes. Figure 4.8 shows control and data ports with associated types for the *Emergency* scenario process from Section 2.2.2. The data port at the output of the *AskWellBeing* Human Task step represents the user's response as Boolean value

(*fine*: true, or *not fine*: false), which is evaluated in the following *If* process step in order to either call an ambulance or terminate the process.

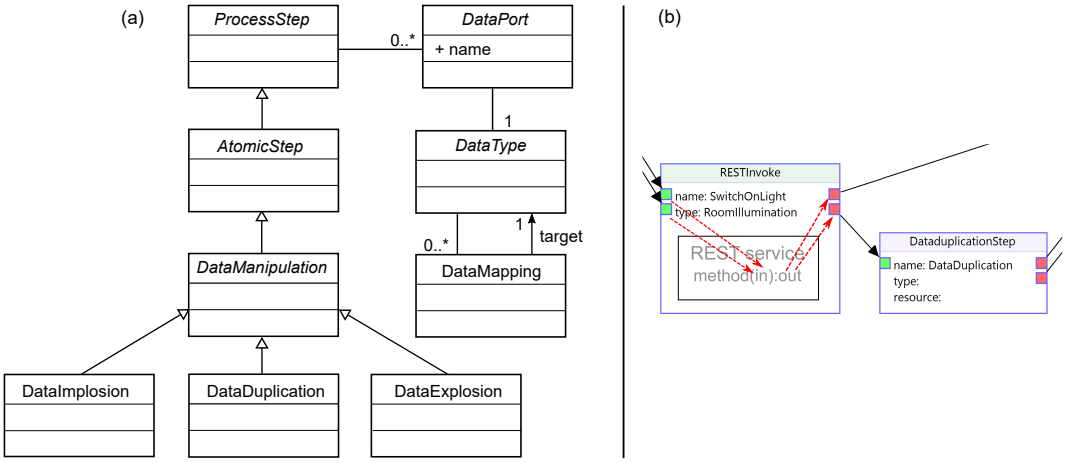


Figure 4.7.: (a) Meta-classes for Data Flow Modelling. (b) Data Flow on Process and Service Level.

In addition, mappings between specific (sub)data instances to data instances of connected process steps' ports can be specified. Figure 4.7(a) shows the process metamodel elements used to define typed data ports for process steps and mappings between data types. In Figure 4.7(b) the flow and mapping of process data to input data of a REST service call (specialization of *ProcessStep* class) and the re-mapping of the service response is presented. This mapping is done automatically based on metadata in the process model (e. g., the name of the data port) and the corresponding attributes (names) of the service parameters in the XML or JSON-based service description.

Figure 4.7(a) also shows extensions of the atomic process step to perform additional manipulation of data: the *DataImplosion* process step allows for combining data from several ports into one data instance; the opposite–decomposing data from a complex data instance to several ports–is achieved by executing the *DataExplosion* step; and the copying of data instances from one port to other ports is modelled using the *DataDuplication* step (cf. Figure 4.7(b)). More complex binary data can be integrated into processes by its resource identifier on local or remote storage.

4.2.7. Escalation Ports, Failure Ports and Failure Branches

Besides *Control Ports* and *Data Ports*, the process metamodel includes special elements for error and exception handling. *Escalation Ports* can be used to define a certain time frame for a process step after which this port should be activated automatically. This mechanism is useful in case time-critical behaviour is required for a process step, e. g., when waiting for a response from a human in an emergency situation as shown in the example process model in Figure 4.8. *Failure Ports* will be activated in case the process engine recognizes errors during the execution of the respective process step. Both Failure Ports and Escalation Ports can be attached to a process step only once. The corresponding failure handling steps have to be

modelled explicitly as a failure branch connected to the respective port. Upon activation of a failure or escalation port, the main outgoing process branches attached to the escalated/failed process steps are deactivated.

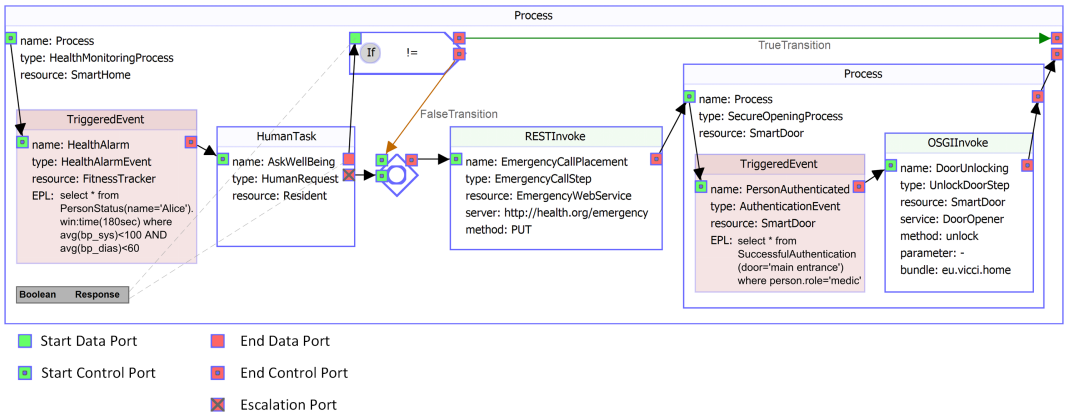


Figure 4.8.: Emergency Scenario Process Model with Control Flow, Data Flow and Escalation Port in our Graphical Modelling Notation.

4.2.8. Services

SOAs are the predominant paradigm to couple heterogeneous systems and applications in a loosely manner [Pap03] (cf. Section 2.3.5). The main focus of current workflow systems is to provide flexible ways of orchestrating these services across application and system boundaries (cf. Section 1.1). Many existing workflow engines and related work base the integration of sensors and actuators in the context of IoT and CPS on a service-oriented approach (cf. Section 3.3, [MRM13, GKGK16]). By extending the *Atomic Step* with several subclasses for service invocations, we are able to easily connect external system functionality via standardized or proprietary service calls. As shown in Figure 4.9, subclasses for the invocation of the more common standardised service interfaces REST (*RESTInvoke*), SOAP (*SOAPInvoke*) and Extensible Markup Language Remote Procedure Call (XML-RPC) (*XMLRPCInvoke*) as well as for more proprietary services based on OSGi (*OSGiInvoke*) [All03] and ROS (*ROSInvoke*) [QCG+09] exist and can be easily extended to support other services. RESTful services are the current standard technologies for programming the IoT whereas SOAP services are predominant in more heavy-weight business applications [GIM11]. These specialized process steps are the main building blocks for invoking external functionality on the lowest level of granularity assumed for our workflows (i. e., on the level of single service methods). Attributes of service invocation process steps usually include a Uniform Resource Identifier (URI), a specific service or method to call as well as input and output parameters, which are modelled as data ports of the process step. The concrete service call is then created with the help of these attributes and its response is mapped to the specific outgoing data ports. The modelling of these process-based service invocations can be supported by loading and automatically parsing a document containing a formal description of the respective service (e. g., a WSDL or WADL file) from which the service parameters and input/output ports are generated automatically. In Figure 4.8 two service calls

are modelled as part of the emergency process: a REST call for invoking an external emergency service in the Cloud; and an OSGi call for activating the door actuator to open the entrance door. In case the functionality to be invoked via a process is accessible only locally and not through a service-based interface, a *LoadClassStep* exists as another specialization of an atomic process step. This process modelling element allows for loading arbitrary Java classes and invoking the desired methods using the Java Reflections Application Programming Interface (API).

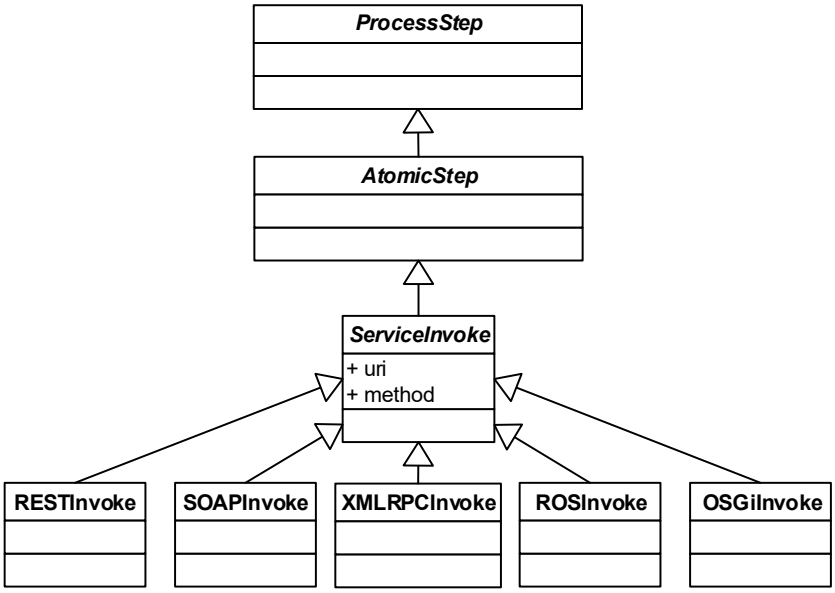


Figure 4.9.: Metamodel Extensions for Service Calls.

4.2.9. Events and Event Abstractions

The concept of events in workflow languages evolved from EPCs as proposed by Scheer within the *ARIS* framework [Sch13] to a large set of different event types in the BPMN 2.0 specification and its application in service-oriented workflow engines [SVDS12]. Events from various physical and virtual sensors and other sources also play an important role in CPS [Tal08]. Section 2.6 discusses the integration of sensors and actuators as a key requirement to manage workflows in CPS in detail (Requirement *R1*). As concluded from the discussion of related work in Section 3.3, the modelling of single sensor sources (e. g., as proposed by Domingos et al. [DMC14] or Meyer et al. [MRM13]) is not feasible for large scale sensor and event networks that make up CPS. Therefore, we rely on a pattern-based approach to evaluate possibly large event streams using CEP for sensor event abstraction [SSOK13] and processing as proposed by Baumgraß et al. [BBDC+15].

In our metamodel, an *Event* is an extension of an atomic process step (cf. Figure 4.10(a)), i. e., it is modelled as a process step containing input and output ports, which are connected to other process steps. An EPL statement is used for defining the activation pattern of a *TriggeredEvent* process step, i. e., an event triggered by external events and *consumed* by the process instance. This process step is activated once all its ingoing ports are active. Then a listener for the defined event

(EPL) pattern is registered and the execution waits and analyses event data with respect to the pattern. Once the pattern can be detected, the outgoing ports of the *TriggeredEvent* step are activated and the execution continues (cf. Section 5.2.4).

The Structured Query Language (SQL)-like EPL provides a very expressive syntax to describe patterns within event streams [Luc02]–from simple ECA rules to complex activation patterns based on temporal conditions, event combinations or arithmetic operations. These patterns can be defined on the level of single event instances with respect to low-level sensor events to aggregated events on a higher level; on the level of event types; and also depending on multiple events of different types. Compared with simple ECA rules and the large set of ambiguous events existing in BPMN 2.0, EPL statements are more expressive and suitable to be used in CPS [BCD+15]. However, the specification of these events is rather complex and requires domain knowledge as well as advanced knowledge of the EPL syntax and available events and their attributes.

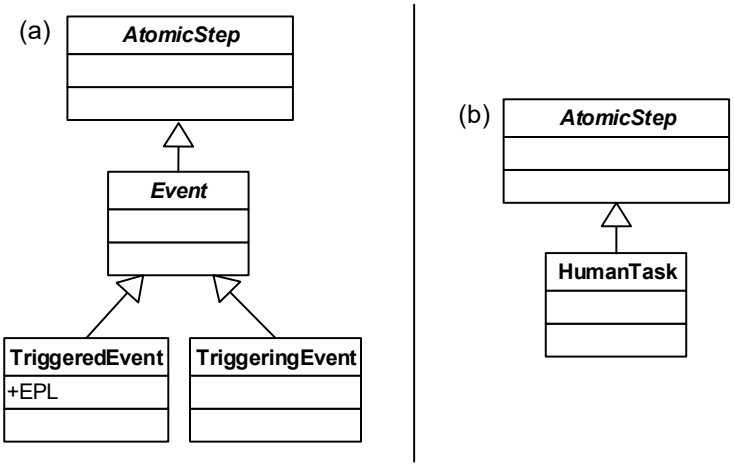


Figure 4.10.: (a) Event Extension and (b) Human Task Extension of Atomic Process Steps.

The *TriggeredEvent* process step in Figure 4.8 contains an EPL statement for the emergency scenario process, which defines the activation (triggering) of that particular event if the sensor events from the health monitor regarding the blood pressure of person *Alice* will drop below 100 mmHg systolic and 60 mmHg diastolic on average within 180 seconds. Once this process step is activated, a listener for this particular EPL statement is created. The event and its outgoing ports become active if the pattern is detected by the CEP engine within the event cloud (cf. Section 5.2.4). This triggered event is specified on the level of the process model, but it refers to a particular instance of a person, i. .e, there has to be a process model dedicated to each resident of the AAL-enabled smart home. The EPL patterns and related data models do not impose restrictions on the properties of an event, though. Events can also be defined on the type level given the data model and intention of the respective process allow such a specification. The specific event payloads that led to the activation can be transferred to succeeding process steps using data ports and event names as matching criterion for the mapping between event and data port.

Using the *TriggeringEvent* class, events *produced* by the process instance and inserted into the event cloud are defined. Once all of these process step's ingoing ports are active, an event is emitted having the ingoing data port's data instances as payload. This higher level event can then again be used as part of an event pattern defining the activation of a *TriggeredEvent*. The data model that defines event types used by the CEP engine is derived from the *Knowledge Base* (cf. Section 4.3).

4.2.10. Human Tasks

The integration of and interaction with humans is discussed as an important requirement to expand CPS towards *Ubiquitous Systems* [Wei91] in Section 2.6 (Requirement *R3*). To integrate human activities into workflows, we extend the metamodel's atomic step with a special *Human Task* process step (cf. Figure 4.10(b)) in analogy to the *WS-HumanTask* specification from the *WS-BPEL4People* extension [KKL+05, AAD+07]. A human task represents an activity within the process that needs to be executed by humans, e. g., to provide data, to react to errors, or simply to confirm the execution of a manual task so that the process instance can continue. The data at the human task's ingoing ports as well as its description should be presented to the user in order to communicate the task at hand. A suitable way for entering required data at the outgoing data ports should be offered to the user (cf. Section 5.6.2). A control port at the end of the respective human task would suffice in case a simple confirmation that a task has been completed by the user is needed. Figure 4.8 shows a *HumanTask* process step within the emergency scenario process used to enquire the resident about her/his health. The process step's outgoing data port is of type Boolean representing her state of health depending on her answer (*fine* or *not fine*). A simple client application presents this human task including its description and an option to answer the question on her mobile end-user device (cf. Section 5.6.2).

4.2.11. Summary

Thus far, a metamodel for describing complex workflow structures including control flow and data flow has been introduced. The complete process metamodel can be found in Section C. This metamodel can be used to define executable workflows for CPS with respect to requirements *R1–R4* identified in Section 2.6. The *Emergency Process* example scenario presented in Figure 4.8 shows the application of the metamodel to define a complex CPS workflow. The focus of the workflow metamodel is on the rather static integration of services, events and humans as basic building blocks of workflows in CPS. However, adding the capabilities of dynamic service selection, cyber-physical synchronisation and self-adaptation/self-management to the list of supported features (Requirements *R2, R5–R8*) for CPS workflows requires more information to be added to the process models. One of the central components of self-managed systems described in the following section is the *Knowledge Base* containing all relevant information that is used for modelling domain specific behaviour and for enabling self-* capabilities (cf. Section 2.4.5).

4.3. Knowledge Base

The MAPE-K feedback loop is one of the central principles from the engineering of self-managed systems field [DLGM+13]. The individual phases of this loop rely heavily on the information contained in a *Knowledge Base* shared among the entities of the CPS. The knowledge base stores the models used to describe the CPS entities and workflows as well as their contexts. It also contains and updates runtime and history information regarding the respective instances. In order to increase the resilience of the WfMS, this information is used within the individual *MAPE-K* phases as shared *Knowledge* for analysis and planning purposes (cf. Chapter 6).

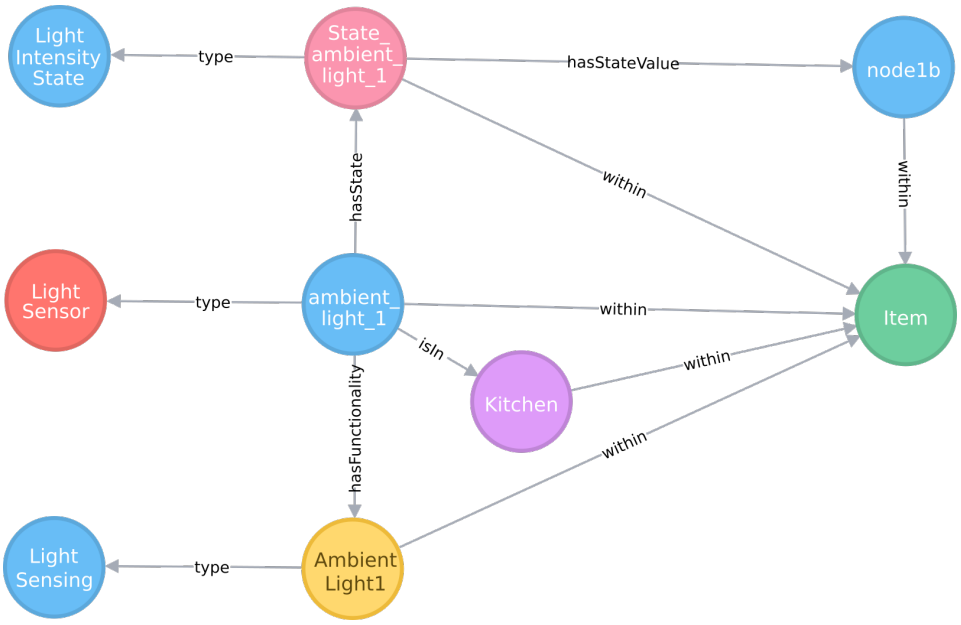


Figure 4.11.: Parts of the Context Model for an Instance of a Light Sensor.

In our work, the knowledge base is founded on the *DogOnt* ontology, which provides an ontological model for describing intelligent domotic environments [BC08, BGT17]. This ontology follows a functionality and capability-based approach for the semantic description of sensors and actuators–including their context–in smart home settings. Therefore, it is suitable to be used as the basic model to describe CPS entities within the scope of this thesis. We combine and apply this ontology to the openHAB middleware[1] and its entities (*Items* as first class citizens). The following sections show the application of the DogOnt ontology to model CPS devices–*Items*–from openHAB (e. g., sensors and actuators) and exemplary extensions that we introduced to model workflow related data as well as more complex combinations of sensors and actuators in the form of robots and humans. These models are described in more detail in [HSKS16b, SSAS15]. We decided to use a semantic model (Ontology) as foundation of our knowledge base as it provides a better expressiveness for modelling entities, concepts and their relations among each

[1] `https://www.openhab.org/`

other and it allows for deriving new knowledge via inference, which will be useful for future extensions of the system [KKMZ17].

4.3.1. Sensors

Sensors are vital elements of CPS and IoT. Figure 4.11 shows details of the sensor model graph in the knowledge base for an instance of an ambient light sensor (*ambient_light_1* node). Besides its type (*Light Sensor*), the knowledge base contains a URI and a unique identifier; its context (i. e., physical location *Kitchen* and associated location coordinates within a map of the building); its functionality and type (*Light Sensing*); and its state including the concrete value for the current luminance of this sensor instance (*node1b*). The *Type* classes associated with a sensor represent the physical values being measured by the respective sensor. They are also the basis for the event data model applied within the CEP engine (cf. Section 4.2.9).

4.3.2. Actuators

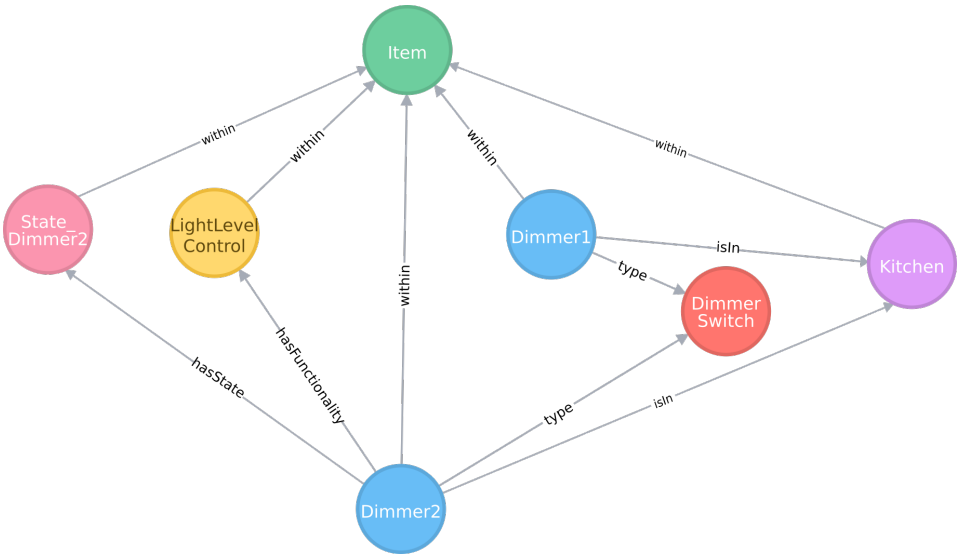


Figure 4.12.: Parts of the Context Model for a Dimmer Switch.

In analogy to sensors, actuators are also represented within the knowledge base as shown for two instances of light dimmer switches (*Dimmer1* and *Dimmer2*) in Figure 4.12. The *State* node represents the current state of the light switch (i. e., its power level in percent). Its *Light Level Control* functionality refers to light level control commands as nodes (*On*, *Off*, *Up*, *Down*), each of which can be invoked via a specific service URI. These states and commands are derived from openHAB's internal model for actuators. The *Light Level Control* functionality also has a relation to the *Light Intensity* node. This relation is used to connect actuators to the physical values they are able to manipulate and therefore to also connect them to the respective sensors. Listing 4.1 shows the textual representation of an instance of a HomeMatic KeyMatic door actuator in Resource Description Framework (RDF) notation for semantic data following the *Manchester* syntax [HDG+06, HSKS16b].

Listing 4.1: Instance of a HomeMatic KeyMatic Door Opener in RDF Description.

```
instance:Thing_keymatic_1
 rdf:type dogont:DoorActuator;
 dogont:hasFunctionality instance: Function_Keymatic_1;
 dogont:hasState instance:State_Keymatic_1;
 dogont:isIn instance:Lobby_0 .

instance:Function_Keymatic_1
 rdf:type dogont:OpenCloseFunctionality .

instance:State_KeyMatic_1
 rdf:type dogont:OpenCloseState;
 dogont:hasStateValue [ rdf:type dogont:OpenStateValue;
        dogont:realStateValue "CLOSED"^^xsd:string ] .

instance:Lobby_0
 rdf:type dogont:Lobby;
 rdfs:label "Lobby"^^xsd:sring .
```

4.3.3. Robots

The previous sections describe how to integrate relatively simple sensors and actuators into the knowledge base used in the context of this thesis. Robots are currently on the verge of becoming ubiquitous assistants throughout almost all domains and phases of everyday lives [SG07]. They are perfect examples of more complex combinations of sensors and actuators, which interact with each other and require more abstract models and levels of detail to be represented in the knowledge base.

Robots are usually composed of several subdevices–either sensors, actuators or computational units–which can be represented as individual entities (*Items*) belonging to the robot's context in the knowledge base. Figure 4.13 shows the example of a TurtleBot service robot having several subcomponents (e. g., a Kobuki locomotion engine, an Asus Xtion camera and a control laptop). The robot has a set of runtime metrics (*RTMetric*) including the power/battery level and a liveliness timestamp (heartbeat) as well as its current location based on a map of the respective building. As the robot may also be able to host an instance of the workflow engine, it can have several (sub)processes running on its computing unit. A robot's functionality can become very complex and specific for that robot type. The level of detail necessary to describe all properties and functionality may become to fine grained and implementation-specific to be able to support a wide range of service robots. In addition, the composition of workflows involving service robots or the development of robotic applications may also be hindered by the complexity and level of detail of current robot control software (e. g., ROS [QCG+09]). Therefore, the knowledge base contains the *hasCapability* relation to have a more abstract and general view on a robot's capabilities–with a service robot being an example of a complex combination of sensors and actuators. In Section A, we present the *DROiT API*–an approach of a capability-based abstraction for the class of small domestic service robots as described in [SSAS15]–and its underlying models for implementing the abstraction for concrete service robots and the capabilities shown in Figure 4.13.

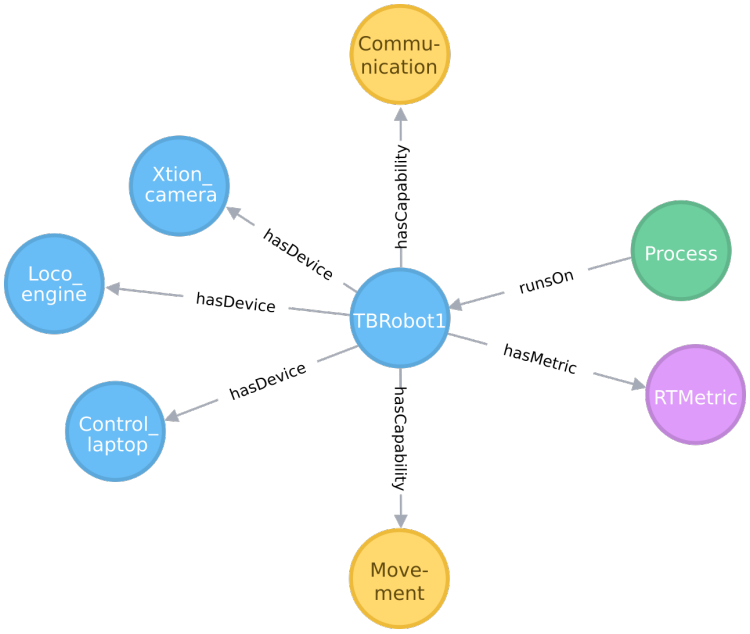


Figure 4.13.: Parts of the Context Model for a TurtleBot Robot.

The aforementioned models based on the *DROiT API* are one example of abstracting and integrating more complex CPS devices into the knowledge base. The capability-oriented approach provides a more abstract view on the robots than the functionality-based approach, which is why it can be applied to a wider range of service robots. Application development and workflow integration are simplified as many implementation and technological details regarding the offered capabilities are hidden behind the API. Capabilities integrate well with the knowledge base providing an additional view on the CPS devices. We have used the *DROiT API* within various workflow related use cases and CPS applications [SLSS16, SNS14a, SHA17, SNK+15]. Besides an easier integration and support of a wider range of robots–due to it being on a more abstract level–the capability-based abstraction also facilitates the dynamic resource selection at runtime. Applications and workflows can request a suitable device based on required capabilities (e. g., a robot able to fetch things), which may result in more resources available and able to execute the specific task than using functionalities or other properties to specify resource requirements.

In general, the simplification of complex functionality CPS devices provide to a more abstract view is a necessary step to integrate these devices into the knowledge base and to use their functionality and capabilities on the workflow level. In our approach, we represent the complex device as well as its set of possible subcomponents as main entities. The functionality of the device and with that, its capabilities vary depending on the availability of certain components. This abstraction process of viewing complex devices as a composition of smaller devices, which add functionalities and capabilities to the overall system can also be applied to other CPS devices (e. g., production machines, household appliances or personal devices). CPS devices interact with physical objects and their environment, which requires a certain degree of context awareness and a representation of objects and locations as their physical

contexts in the knowledge base. By also including these concepts (here: *MovementTarget* and *GrabbingTarget* locations) in our abstract models (cf. Section A), we enable their augmentation with additional context information (e. g., the physical conditions at a movement target or the physical location of a grabbing target).

Robots are also good examples to show the complex interactions of sensors, actuators and computing units. Programmed processes run on various levels of the hardware platforms: within single devices (e. g., microprocessors controlling specific motors or processing camera data), single systems (e. g., computers controlling the entire robot), or across system boundaries (e. g., workflows orchestrating multiple devices in the smart home) (cf. Section 2.5.1). The focus of this work is on the latter level of workflows among systems and systems of systems (PACPS). Functionalities and capabilities of a device or a system have to be accessible via service-based interfaces in order to be included in a process model. The invocation of a service from the workflow layer may trigger internal processes on the device or system level, which are regarded as black boxes and not considered by this work. That way, more (safety-) critical and other safety-related features can be implemented closer to the actual hardware involved in the respective processes. For example, the processes regarding obstacle avoidance can be executed directly on the computer controlling the robot and not necessarily by the workflow engine.

4.3.4. Humans

CPS, IoT and especially ubiquitous systems put a strong emphasis on integrating users into the overall systems, which is why the representation of a person is also needed in the knowledge base (Requirement *R3*: Human Interaction). However, there is only little to no related research regarding the ontological representation of humans, their capabilities and contexts. The *Friend Of A Friend* (FOAF) ontology mostly specifies properties of persons and their relations among each other [GCB07]. The *SOUPA* ontology adds more properties as well as Beliefs, Desires and Intentions (BDI) from agent-based systems to the models to describe goals and plans of people [CPFJ04]. These works focus on humans as central entities of the whole system. In our approach, people are considered as first class citizens of the CPS along with the aforementioned sensors, actuators and more complex entities. In analogy to sensors and actuators described in the previous sections, the *Person* class in the knowledge base is also linked to context factors (e. g., a location) as well as to capabilities the individual instance of a person has (e. g., grabbing, movement or communication from the robot example in Section A). The SOUPA ontology could be used to extend the information in the knowledge base with respect to humans, their behaviour and intentions in the CPS.

4.3.5. Workflows

Workflow-specific data is contained in the knowledge base according to the descriptions in previous sections of this chapter. This information comprises process (step) and instance attributes as well as the *runsOn* relation, which links a (sub)process instance to its executing device (i. e., resource) as shown in Figure 4.13. Additional process-related data concerning the physical effects of activities will be introduced in Section 4.5. *Goals* and *Objectives* will be used to describe and verify the expected

outcome of the process execution, to detect exceptions and undesired behaviour, and to adapt the process in case of errors.

4.4. Dynamic Services

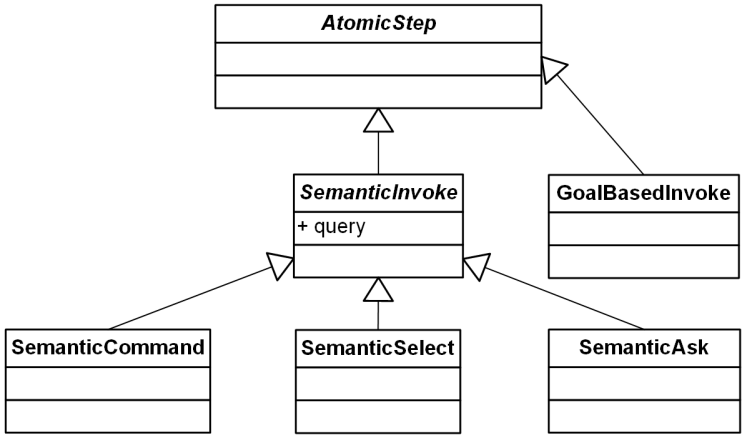


Figure 4.14.: Extension of the Workflow Metamodel for Dynamic Service Selection.

Thus far, components within CPS can only be accessed based on rather static IoT service address definitions and requests (cf. Section 2.4.4). As pointed out in Section 2.6 (Requirement *R2*: Dynamic Resources), the availability of devices and resources within CPS may vary heavily over time, making static service calls infeasible and error prone [CSOL15]. Therefore, the workflow metamodel also supports special process steps to enable a more general (under-)specification of workflow activities as described in [HSKS16b]. This specification relies on semantic SPARQL Protocol And RDF Query Language (SPARQL) queries to be executed on the underlying ontology. The queries specify necessary functionalities, capabilities and context constraints of resource instances available from the knowledge base (cf. Section 4.3) to execute the respective workflow task. As suggested in [DBBM11], a semantic model to describe the components of an IoT environment is feasible to enable a more dynamic and flexible discovery of resources and services. We distinguish between three types of SPARQL-based knowledge base (ontology) queries as subclasses of a *Semantic Invoke* (cf. Figure 4.14) process step to retrieve specific devices, IoT service parameters associated with the device functionality, or sensor values from the knowledge base:

- **Semantic Select**: This query is used for a one-time retrieval of the current state of actuators or values of sensors in a certain context. Listing 4.2 shows an exemplary query for retrieving the current brightness level in a given location.
- **Semantic Ask**: This query is used for a one-time retrieval of certain data and values (e. g., sensor values) from the knowledge base and evaluating these values based on a given condition. Listing 4.3 shows an exemplary query for checking if the brightness levels in the current location are below 290.1 Lux.

Listing 4.2: Semantic Select Query for Retrieving Current Luminance Levels.

```
SELECT ?realLoc ?curRealLoc ?lightState ?currentValue
WHERE {
   instance:State_Current_Location dogont:hasStateValue ?
      stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasState ?lightState .
   ?lightState dogont:hasStateValue ?lightValue .
   ?lightValue rdf:type ?type.
   ?type rdfs:subClassOf* dogont:BrightnessStateValue .
   ?lightValue dogont:realStateValue ?currentValue .
   ?thing dogont:isIn ?loc .
   ?loc rdfs:label ?realLoc .
FILTER(?curRealLoc = ?realLoc)}
```

Listing 4.3: Smenatic Ask Query for Checking Current Illuminance Levels.

```
ASK
WHERE {
   instance:State_Current_Location dogont:hasStateValue ?
      stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasState ?lightState .
   ?lightState dogont:hasStateValue ?lightValue .
   ?lightValue rdf:type ?type.
   ?type rdfs:subClassOf* dogont:BrightnessStateValue .
   ?lightValue dogont:realStateValue ?currentValue .
   ?thing dogont:isIn ?loc .
   ?loc rdfs:label ?realLoc .
FILTER(?curRealLoc = ?realLoc && xsd:double(?currentValue
   ) < 290.1)}
```

- **Semantic Command**: This query is used for finding certain types of actuators in a specific context. The execution of an instance of this process step leads to the invocation of the respective functions as defined in the query for all instances of actuators matching the query criteria. Listing 4.4 shows an exemplary query for finding all actuators of type *Dimmer Switch* in a given location able to execute the *OnCommand*.

In addition, we support the specification of a goal within a *GoalBasedInvoke* process step for defining the outcome of a certain process and additional non-functional properties using the *TROPOS* specification methodology [BPG+04]. These specified goals are evaluated and partially converted into the SPARQL queries described above [HSK+16]. More detailed elaborations by Huber on applying goals and the TROPOS methodology for dynamic service discovery and workflow adaptations for role-based resources in the IoT can be found in [Hub18]. These more abstract specifications of properties and context constraints that need to be fulfilled by process resources in order to execute the respective tasks enables a more flexible allocation

Listing 4.4: Semantic Command Query for Retrieving Dimmer Actuators.

```
SELECT ?func
WHERE {
   instance:State_Current_Location dogont:hasStateValue ?
      stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasFunctionality ?func .
   ?thing rdf:type ?thingType .
   ?thing dogont:isIn ?loc .
   ?thingType rdfs:subClassOf* dogont_DimmerSwitch .
   ?func dogont:hasCommand ?cmd .
   ?cmd rdf:type ?cmdType .
   ?cmdType rdfs:subClassOf* dogont:OnCommand .
   ?loc rdfs:label ?realLoc .
FILTER(?realLoc = ?curRealLoc)}
```

of resources at runtime. Available resources and their specific service addresses do not have to be known at workflow design time. They can be discovered and invoked dynamically based on the presented queries, model and instance information contained in the knowledge base. In addition, the complexity of process models can be reduced as not every service invocation has to be modelled as an individual process step for multiple devices or services. The results of the semantic invoke extensions may comprise the querying and triggering of multiple sensors and actuators that match the specified criteria. That way, we can for example trigger all lights in a specific room to be switched on without requiring a priori knowledge about available light sources and without the specification of the individual process activities to switch on each individual device.

4.5. CPS-related Workflow Effects

Thus far, we are able to specify ordered sequences of process steps and activities to be executed as part of CPS workflows. These workflows may include–besides logic and other elements for manipulating the control flow–interactions with static and dynamic services, actuators, sensors and humans. The meta-classes and attributes presented in previous sections are used to define active and reactive conditional actions including control flow and data flow similar to other high-level workflow languages. In order to verify the successful execution of the individual activities or detect the occurrence of exceptions and errors, we also need a way to specify success and error criteria regarding the expected outcome and effects of executing a process step in the physical world and in the virtual world.

4.5.1. Metamodel Extensions

One of the major differences of CPS workflows compared to “traditional” business workflows is their ability to also influence the physical world. As pointed out in Section 2.5, these physical effects have to be taken into consideration when executing an instance of a CPS workflow. The synchronization of the digital model of the

physical world with the actual real world states (*Cyber-physical Synchronization*) to achieve *Cyber-physical Consistency* (cf. Section 4.6) is an important requirement for engineering resilient CPS workflow management systems (Requirements *R5–R7* in Section 2.6). However, current workflow languages do not support the specification of the relevant CPS aspects regarding the physical effects and outcome of processes. Therefore, we propose an extension of the previously described metamodel to also specify these effects as described in [SHHA16, SHHA17]. These extensions can also be applied to other workflow modelling languages such as BPMN 2.0 or WS-BPEL.

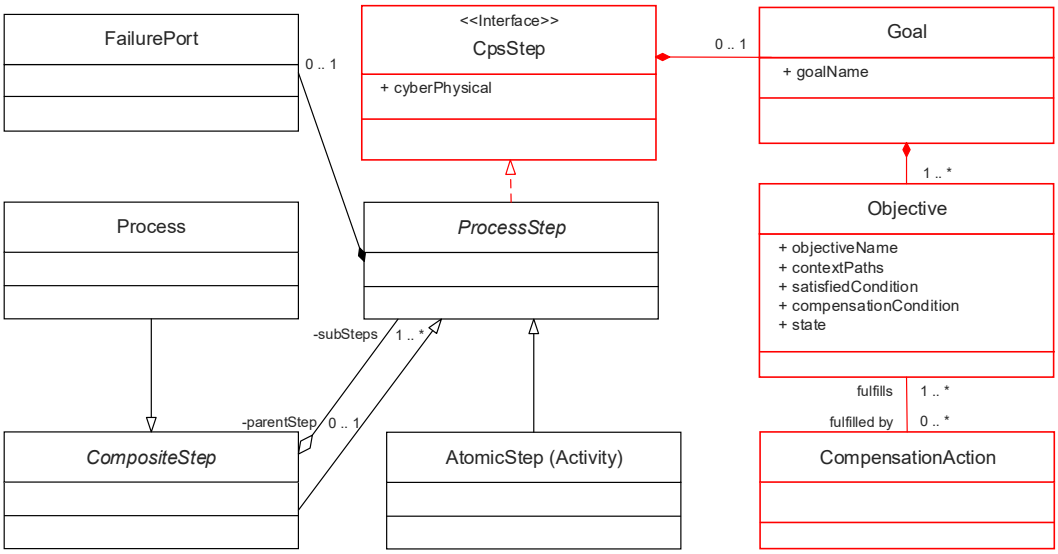


Figure 4.15.: Metamodel Extensions for Specifying Effects/Success Criteria in Objectives for the Workflow Execution regarding CPS Aspects.

Figure 4.15 shows the proposed extensions for the process metamodel described in Section 4.2 in UML notation. Process steps–either composite process steps or atomic process steps (*Activities* in BPMN 2.0)–have to implement the new *CpsStep* interface, which adds the *cyberPhysical* property to the respective process step. This attribute indicates if the respective process step influences the physical world. In case this is true, a *Goal Go* can be specified to define the domain-specific outcome, effects regarding relevant context factors, and success and error criteria for the process step as proposed in [KK12]. This more declarative goal-oriented approach is especially suitable for complex SoS, because modelling every possible outcome and error as well as possible compensations for the process step in question and associated failure handling process branches is not feasible [Kop13, MPMR16]. The *cyber-physical* flag is used in combination with the goals to decide about the selection of appropriate compensation actions in the MAPE-K loops for exceptions that occurred (cf. Section 6.3). Regarding the error compensation for cyber-physical process steps, we try to find replacement actuators in the same context as the *original* process resource and invoke the respective commands to execute similar functionality.

The *Goal* property $Go = (gn, O)$ consists of a Goal Name gn and a set of *Objectives* $O = \{o_1, \ldots, o_n\}$ that need to be fulfilled to confirm the successful execution of the process step and ensure cyber-physical consistency CPC (cf. Definition 4.1). An exemplary goal Go of a process activity or subprocess could be gn ='provide enough

Listing 4.5: Query Specifying the Context Path to a Specific Light Sensor.

```
MATCH (kitchen)-[:instanceOf]->(room)
MATCH (light)-[:instanceOf]->(sensor)
MATCH (light)-[:isIn]->(kitchen)
RETURN light.value AS lightIntensity
```

light for working in the kitchen'. This goal includes the objective o_1 for the process step to increase the light levels in a certain room (here: kitchen) above a certain threshold (here: 700 Lux) within a certain time frame (here: 5 seconds). In case one of the objectives cannot be fulfilled, an error is assumed and a *CompensationAction* for this error has to be searched for (Requirements *R5* and *R6*). The compensation action is derived in the *Plan* phase of the MAPE-K feedback executions based on the occurred mismatch and suitable compensation queries from a compensation repository (cf. Section 6.2). A compensation action is able to fulfil one or more objectives. On the other hand, multiple compensation actions may be necessary to fulfil an objective or an objective cannot be fulfilled by compensation actions at all. An objective $o_i = (on, CP, sc, cc)$ contains an Objective Name on and the following attributes (cf. Figure 4.15).

Context Paths

The context paths $CP = \{cp_1, \ldots, cp_n\}$ specify paths cp_i to relevant context attributes $c_i \in S_{P,t}$ of the actual state as measuring points to be monitored during the execution of the process step. These measuring points can be sensor data, process-related events or other context factors that are influenced by the process (cf. Section 4.6). As there is no need for using inference or semantic data at this point, the information contained in the knowledge base (cf. Section 4.3) is viewed, stored and processed based on a graph $G = (V, E)$ with the vertices V being entities within the knowledge base (here: graph database) and the edges E being relations between these entities. With the large number of interconnected devices and entities that make up CPS and the IoT, graphs and graph databases are usually more feasible, expressive and faster than classical object relational databases or semantic triple stores when processing data and evaluating relationships at runtime [RWE15]. A context path cp_i is a path along the edges $(x_1, \ldots, x_n) \in E$ within the graph G (representing the structure of the CPS) specified using a graph query written in the *Cypher* graph query language for *Neo4j* databases [Web12], which are the technological foundation of the graph database used at this point. The graph query represents a path pointing to a specific vertex $c_i^{'} \in V$, which represents data nodes (context values) in the knowledge base. An exemplary context path cp_i leading to the value of a light sensor node $c_i \in S_{P,t}$ (*lightIntensity*) in the kitchen written as Cypher query is shown in Listing 4.5.

Satisfied Condition

The satisfied condition sc defines the criteria for fulfilling the particular objective o_i with the left side referring to context attributes $c_{i,t}$ at time t from the physical

Listing 4.6: Satisfied Condition for Successful Light Switching.

```
1 #lightIntensity > 700
```

Listing 4.7: Compensation Condition for Erroneous Light Switching.

```
1 #objective.created.isBefore(#now.minusSeconds(5))
```

state $S_{P,t}$ as addressed by the respective context path cp_i and the right side defining the corresponding assumed/expected context attribute $c'_{i,t}$ from the virtual process state $S_{C,t}$ as target value. If this condition can be evaluated positively, the objective o_i belonging to the particular process step is assumed to be fulfilled. If all satisfied conditions for a goal are true, then cyber-physical consistency is maintained for the respective process step. An exemplary satisfied condition (written in Spring Expression Language (SpEL) [JHA+13]) requiring the light level ($c_{i,t}$) to be above a certain threshold (700 Lux) is shown in Listing 4.6.

Compensation Condition

The compensation condition cc is used to specify a Boolean error/exception criterion, which suggests that the execution was not successful and an error occurred (when *true*). This condition may include context data c_i from the physical state $S_{P,t}$ specified in the context paths $cp_i \in CP$ as well as special functions and additional context data, e. g., time frames or averages. The compensation condition is used to indicate the need for finding a compensating action to handle the occurred error (Requirement *R6*: CPS Errors). An exemplary compensation condition written in SpEL testing if an objective was created more than 5 seconds ago is shown in Listing 4.7. We assume that after a maximum of 5 seconds there should be a significant change within the light levels, otherwise there is a malfunction regarding the light switch actuator.

State

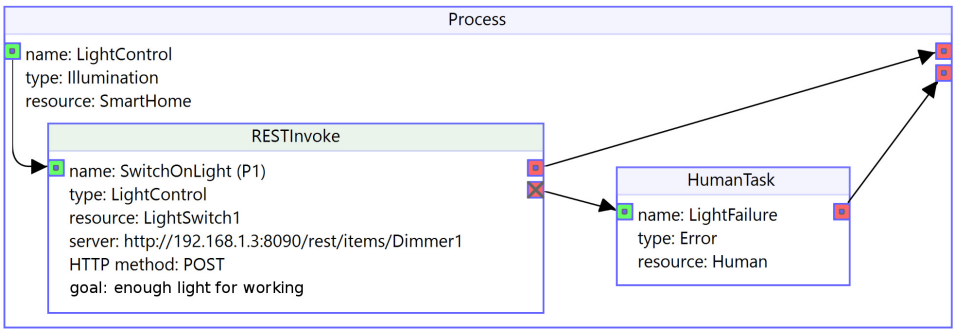


Figure 4.16.: Example Process with Failure Port and Human Task in Failure Branch.

This attribute adds a runtime state to the individual objective. An objective can be satisfied, unsatisfied, in need of compensation, or failed. If the satisfied condition *sc* can be evaluated positively, then the objective is in state *satisfied*. The need for compensation can be determined by evaluating the compensation condition *cc*. If neither condition is true, the objective is of state *unsatisfied*. In case the objective cannot be fulfilled by the MAPE-K feedback loop, i. e., a compensation is needed but cannot be found, it enters the *failed* state. The overall goal is fulfilled if all of its objectives are satisfied (i. e., all relevant sensor values as defined in the objectives indicate the fulfilment of the goal and therefore the successful execution). At that point, cyber-physical consistency for the respective process step is presumed and the process execution continues. As shown in Figure 4.16, a *Failure Port* may be specified as part of the cyber-physical process step (cf. Section 4.2.7). This port will be activated in case the process step's goal (i. e., at least one of its objectives) cannot be fulfilled (i. e., a compensation for the occurred error cannot be found). This activation will then trigger the manually modelled process branch connected to the failure port to be executed. By adding a dedicated process branch for unresolvable error cases to the failure port, the process designer is able to specify the fallback process behaviour for that specific error case/exception using all concepts of the workflow language. With respect to the smart lighting scenario, this error branch may include a *Human Task*, which notifies the user to check the lamps in the specific room manually in case no compensating action for the broken lights can be found. Figure 4.16 shows an example process that contains the invocation of a REST service to switch on a specific dimmer. This process step is annotated with the goal "enough light for working", which was described in previous sections. In case the goal cannot be fulfilled due to missing compensation actions (e. g., no alternative light switches are available), the process step's failure port is activated and a human task triggered to inform the user. Other BPM systems and notations may be extended in a similar way to use failure branches or other built-in means for error, exception or compensation handling.

Knowledge Base Extensions

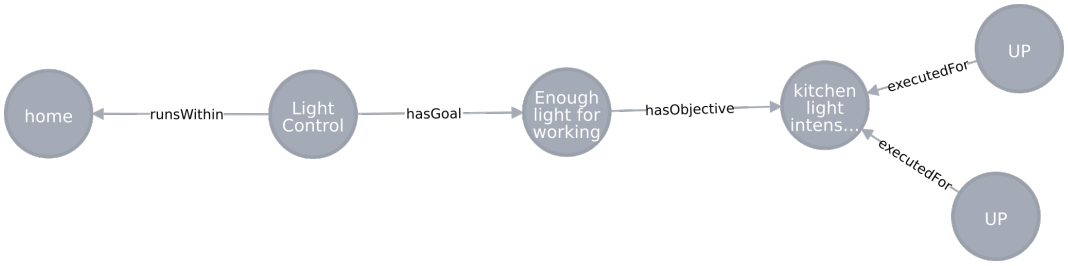


Figure 4.17.: CPS-related Workflow Data in the Knowledge Base.

Goals, objectives and their states are also contained in the knowledge base for further processing. Figure 4.17 shows an extract from the knowledge base with CPS-related workflow data. An instance of the cyber-physical *Light Control* process step runs within the context of the smart home. It has the goal "Enough light for working", which includes the objective of providing a certain level of light intensity as

described in Listings 4.5 and 4.6. The states of the objectives and goals are stored as attributes of the particular nodes. We also store the respective compensation actions that were executed for the specific objective in case of an erroneous process step needing compensations. The example in Figure 4.17 shows that *UP* commands, which are part of the light level control functionality of a dimmer (cf. Figure 4.12) were executed for two dimmers in order to fulfil the objective.

4.5.2. General Applicability of Goals and Objectives

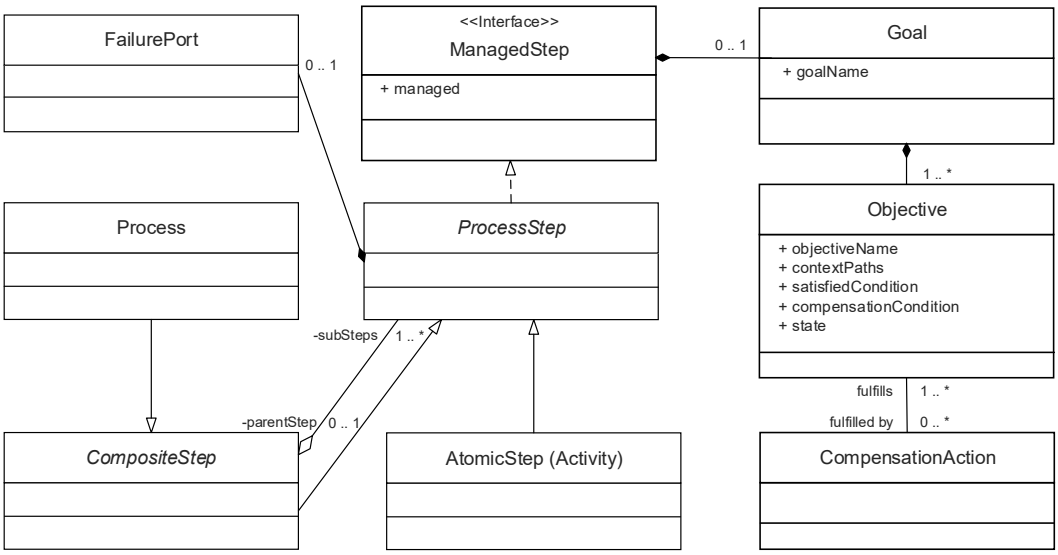


Figure 4.18.: Generic Metamodel Extensions for Specifying the Effects/Success Criteria in Objectives for *Managed* Process Step Executions.

Goals and objectives can also be used to specify non-CPS related aspects and success criteria that need to be fulfilled upon execution of a process (step). These aspects may comprise QoS levels (e. g., throughput or latency requirements for certain operations) [OCEP13], Key Performance Indicators (KPIs) [KK12] or limitations on other physical or virtual context factors (e. g., restrictions on the overall energy consumption of a subprocess [SGCG18]). These metrics have to be contained in the knowledge base of the individual instance of the process management system and they have to be reachable via respective context paths. Due to the composite structure of a process, goals and objectives can be specified on various levels (i. e., on the process level, subprocess level or atomic task level). The process modeller is responsible for specifying these goals in a non-conflicting and non-contradicting way. With goals comprising objectives related to various aspects of the process execution the requirement *R7* of adding the capability of self-adaptation and self-management to processes can be achieved in a sophisticated way. Figure 4.18 shows our generalized proposition of the process metamodel to support the definition of *managed* process steps based on goals and objectives. The *ManagedStep* interface adds the *managed* attribute as well as goals and objectives to the respective process step in analogy with the *CpsStep* interface described in Section 4.5.1. This way, objectives related to virtual context factors and criteria can be specified, and compensation actions for

occurred errors can be derived, too. The selection of suitable strategies for deriving compensation actions depends on the respective attributes of the managed process steps. For erroneous *cyber-physical* process steps, we describe the selection strategy of choosing similar actuators or functionalities as replacements in Section 6.3. For erroneous *managed* distributed subprocesses, we describe the strategy of repeating the execution of the same subprocess on another peer in Section 6.4. General elaborations on the selection process for *managed* process steps can be found in Section 6.2. Depending on the preferred strategies for selecting suitable compensations for self-management of process steps, additional attributes or classifications of use cases and desired behaviour of the MAPE-K loop have to be added to the managed process steps and *Compensation Repository* (cf. Section 6.2).

4.6. Cyber-physical Consistency

A goal *Go* aggregates objectives $o_i \in O$ that define the expected influence of a particular process step on the physical world. As pointed out in requirement *R5*, the aspect of *Cyber-physical Synchronization* is especially important for CPS to ensure a consistent view between the virtual and physical world (cf. Section 2.6). The information contained in the goals is used to check if the process execution was successful and if its outcome is as expected (w. r. t. the satisfied condition *sc*) or–if not–to find possible compensation actions for the erroneous process instance (w. r. t. the compensation condition *cc*). In this work, the notion of *Cyber-physical Consistency CPC* is introduced as an extension of the well-known Atomicity, Consistency, Isolation, Durability (ACID) criteria for distributed systems and databases [GR92].

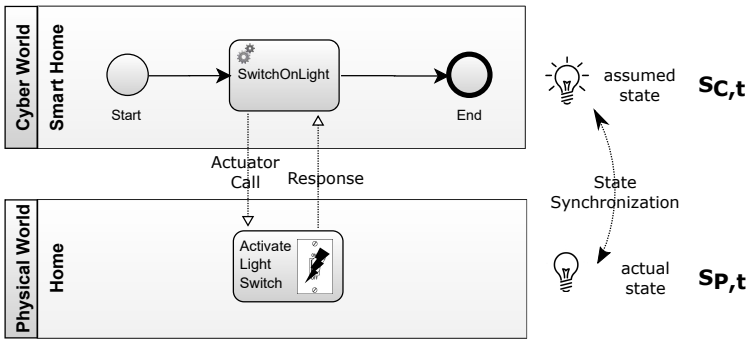


Figure 4.19.: Synchronization Between Cyber World $S_{C,t}$ and Physical World $S_{P,t}$.

In order to verify that the process execution is in a state of cyber-physical consistency, its assumed physical state (*Cyber State* $S_{C,t}$ as defined as target values in the satisfied conditions of process objectives) is compared to the actual real world state (*Physical State* $S_{P,t}$ as measured by sensors) of the entities involved in the process execution as shown in Figure 4.19 for the smart lighting scenario process. Cyber-physical consistency (cf. Definition 3) is reached if both states correspond to each other or if they can be synchronized. It is violated if there is a mismatch between both states. With reaching cyber-physical consistency, we are able to verify that the process execution was successful or detect exceptions/errors. The example shows a simple process issuing a service call to a dimmer actuator to switch on the light in

a certain room in the smart home. A positive service response indicates that the request was executed successfully and the process terminates. However, a broken or worn off light bulb, hardware issues, real world obstacles or wrong parameters may lead to an undesired physical state $S_{P,t}$ of the light switch, which may not be detected by the IoT device's control software or WfMS. In case the service is not able to detect these issues, the cyber state $S_{C,t}$ (*light on*) and process execution state (*successful*) do not correspond to the physical state $S_{P,t}$ (*light off*) and actual execution state (*unsuccessful*), which means that cyber-physical consistency is violated and needs to be restored for that example.

To have a more formal representation of the concept of *Cyber-physical Consistency* (CPC) we denote:

Definition 1 *Physical Process Context* $S_{P,t}$

$C_t = \{c_{1,t}, \ldots, c_{n,t}\}$ *is a set of all physical context attributes* $c_{i,t}$ *of CPS at a given point in time* t*. A context attribute is defined as a tuple* $c_t = (n, v_t)$*, where* $n \in N$ *represents a unique identifier from a set* N *of identifiers and* $v_t \in V_t$ *represents a value from a set* V_t *of context values at a given point in time* t*. We define* $S_{P,t} \subseteq C_t$ *as a set of all physical context attributes that are manipulated by or relevant for the execution of a specific process (step) at time* t *as defined within the objectives* $o_i \in O$ *of the process step's goal Go (Physical Process Context).* $S_{P,t}$ *represents the* ***actual*** *(physical)* ***state*** *of the process execution in CPS at a given point in time* t*.*

Values can be numeric values for measurable factors but also more abstract concepts, e. g., execution states or locations. The physical process context refers to physical factors whose states can be measured by sensors or abstract values depending on the respective sensors (e. g., light=*on*) as addressed by the corresponding context paths $cp_i \in CP$. For our light example, a context attribute $c_{i,t} \in S_{P,t}$ could be the light influenced by the respective process instance identified by n=lightSource1 and having a specific value at time t: v_t=700 Lux.

Corresponding to the previous definition we denote:

Definition 2 *Virtual Process Context* $S_{C,t}$

$C_t^{'} = \{c_{1,t}^{'}, \ldots, c_{n,t}^{'}\}$ *is a set of all virtual representations of physical context attributes* $c_{i,t}^{'}$ *at a given point in time* t *(****assumed state*** *of the physical world). A virtual context attribute is defined as a tuple* $c_t^{'} = (n^{'}, v_t^{'})$*, where* $n^{'} \in N$ *represents a unique identifier and* $v_t^{'} \in V_t$ *represents a value at a given time* t*. We define* $S_{C,t} \subseteq C_t^{'}$ *as a set of all virtual context attributes that are manipulated by or relevant for the process execution (Virtual Process Context).* $S_{C,t}$ *is the* ***assumed*** *(virtual)* ***state*** *of the physical process execution in CPS at a given point in time* t*.*

The virtual process context attributes correspond to the right sides of the conditions referred to in the compensation condition *cc* and satisfied condition *sc* via the corresponding context paths $cp_i \in CP$ used within the objective specifications. The process designer specifies virtual context attributes as *target values* in the satisfied conditions for the execution of individual process tasks or process instances. At

runtime, the values of these virtual context attributes are the values that the computer assumes for the corresponding physical context attributes at a specific point in time t–inconsistencies/deviations from the actual physical context values can be possible.

From the previous definitions we derive the condition for the process execution being in a state of cyber-physical consistency (CPC_t) at a given point in time t:

Definition 3 *Cyber-physical Consistency CPC_t*
For all relevant context attributes $c_t \in S_{P,t}$ (defined in the objectives' context paths) of the actual physical state (cf. Definition 1), there exists a corresponding virtual context attribute $c_t^{'} \in S_{C,t}$ of the assumed state (cf. Definition 2) where $n = n^{'}$ and $v_t = v_t^{'}$ (i. e., the identifier and value of both context attributes are equal).

$$CPC_t \Leftrightarrow \forall c_t : c_t \in S_{P,t},\ \exists c_t^{'} : c_t^{'} \in S_{C,t} \text{ where } c_t \equiv c_t^{'} \text{ with } c_t \equiv c_t^{'} :\ (n = n^{'}) \wedge (v_t = v_t^{'}) \tag{4.1}$$

Analogous to that, we define the condition for the process execution not being in a state of cyber-physical consistency at a given point in time t based on the criteria defined in the corresponding objective:

Definition 4 *Cyber-physical Inconsistency $\neg CPC_t$*
There exists at least one context attribute $c_t \in S_{P,t}$ of the physical state (cf. Definition 1) that has a corresponding virtual context attribute $c_t^{'} \in S_{C,t}$ of the assumed state (cf. Definition 2) where $n = n^{'}$ and $v_t \neq v_t^{'}$ (i. e., the identifiers of both context attributes are equal but their values are different and therefore inconsistent).

$$\neg CPC_t \Leftrightarrow \exists c_t : c_t \in S_{P,t},\ \exists c_t^{'} : c_t^{'} \in S_{C,t} \text{ where } c_t \not\equiv c_t^{'} \text{ with } c_t \not\equiv c_t^{'} :\ (n = n^{'}) \wedge (v_t \neq v_t^{'}) \tag{4.2}$$

So far, the definitions relate cyber-physical consistency CPC_t to a specific point in time t. One or more objectives define the expected target values for relevant context factors from $S_{P,t}$ and $S_{C,t}$ that also refer to that specific time t. Goals aggregate multiple objectives referring to multiple points in time. Hence, goals can be used to specify overall cyber-physical consistency CPC criteria for a certain process activity, subprocess or process referring to multiple points in time.

Robot Movement Example: Figure 4.20 shows a more complex example of multiple objectives specified for the process-controlled movement of a service robot. The overall goal is to move the robot from its docking station to a specific room identified via its x,y-coordinates on the robot's internal map (6,10). The correct starting and end points as well as two intermediate stopping points of the robot's trajectory are defined as individual objectives of the corresponding process step in the order the objectives need to be fulfilled. The fulfilment of these objectives has to be analysed and possibly corrected continuously during the execution for the single process step. The four objectives relate to the specific physical context factors from the *actual* physical state $S_{P,t}$ as target values that represent the robot's location (x,y-coordinates) during the points in time $t_0, .., t_3$. Here, a specific point in

time t does not necessarily need to be an actual date but it can also be related to a particular sensor event (e. g., when the robot passed a light barrier or reports its arrival). From the figure, we see that during the actual movement of the robot, an inconsistency between the physical state $S_{P,t}$ and the cyber state $S_{C,t}$ occurred due to an incorrect positioning of the robot at time t_2 in the physical world. The robot assumed the correct position (4,6) and stopped, publishing an "arrived" event. The corresponding objective's satisfied condition defines successful process execution as the robot emitting an "arrived" event and an external tracking system confirming the correct physical location (4,6). The compensation condition indicates an inconsistency and error during process execution when the robot emits the "arrived" event and the external coordinates not matching the specified target coordinates. The compensation condition became true for the third objective at time t_2 as the external system measured other coordinates (4,8). Therefore, the corresponding objective was not fulfilled and with that, the goal is left unsatisfied and the overall cyber-physical consistency is violated. Although the robot reached its correct final position at time t_3, one of the objectives is unsatisfied. We rely on the process designer to specify reasonable and important objectives for the individual process steps, which is why we consider all objectives as equally important. Therefore, the unsuccessful fulfilment of one objective leads to a violation of the overall consistency. Optional or weighted objectives and also more advanced concepts, e. g., regarding *Eventual (Cyber-physical) Consistency* [Bur14] have to be discussed as part of future investigations. One of the major goals of this thesis is to detect inconsistencies during process execution as described in this example (Requirement *R5*: CPS Sync) and to "repair" the inconsistency before the process execution continues (Requirement *R6*: CPS Errors). We aim at detecting and repairing these inconsistencies at the workflow level to remedy missing functionality, issues or errors, and imprecisions related to the individual process resource (i. e., CPS entity, here: robot). An alternative to specifying the movement targets of the robot by four objectives in one goal for one process step and checking this goal continuously during execution, would be to model four individual process steps for driving to the individual targets and defining the goals to contain one objective each per process step, which is checked during the execution of the individual process step.

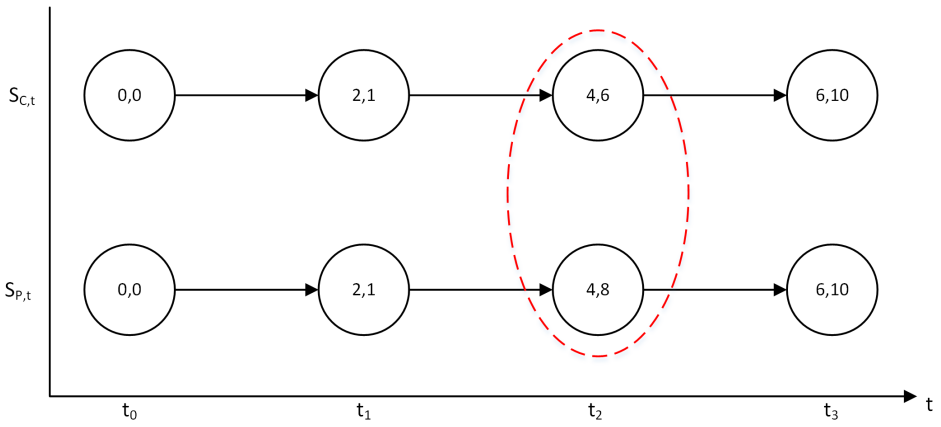


Figure 4.20.: Inconsistent States of the Movement Path Coordinates of a Service Robot controlled by a Process.

CPS workflows are abstract virtual concepts that do not have a physical equivalent. The execution of CPS workflow instances can be traced through their effects on the physical world, i. e., changes within context attributes, states and other properties of the physical environment, of things and objects, or of humans. The parts of a workflow that are able to modify these factors are related to tasks involving sensors and actuators, which are the interfaces to interact with the physical world (cf. Figure 4.19). Other workflow elements, e. g., split and merge operators or the concepts of processes and subprocesses cannot be directly synchronized to physical representations. The extension of the workflow meta-levels depicted in Figure 4.1 into the physical world and introduction of a new *InstanceOf* relation to represent the virtual workflow instance in the physical world is therefore only partially possible or reasonable–mostly related to service invocation tasks to call physical actuator commands. As proposed with the newly introduced concept of *Cyber-physical Consistency* (cf. Definition 3), the evaluation of relevant context values and states in the physical world that were influenced by a workflow instance presents a reasonable way of correlating the physical effects of the process execution with the virtual model/representation of the process (instance). We will elaborate on the relation between CPS workflows, cyber-physical consistency and cyber-physical objects/digital twins as well as on cyber-physical transactions and ACID criteria for CPS workflows in Sections 6.10 and 6.9.

Consistency Levels and Ranges

Depending on the application domain, maintaining a strict level of cyber-physical consistency CPC regarding all relevant physical context attributes from S_P is not always necessary or feasible–especially in the physical world where a lower level of precision is usually sufficient for tasks to be completed than for processes in the virtual world. While, for example within smart factories a high level of precision is required to yield a high product quality, context factors and other criteria defined in goals and objectives for smart home environments can be in a certain range or below/above certain thresholds to achieve the desired levels of comfort and qualities (e. g., with respect to the room temperature or illumination). As shown in the exemplary objectives, we support the definition of conditions for success and error criteria in a more fuzzy way based on comparison operators to specify–besides equality–thresholds and ranges for measurable (numerical) context attributes as applied in classical control theory.

This is complemented by an optional *Consistency Level* L_o for an objective o:

Definition 5 *Consistency Level* L_o

The Consistency Level L_o *can be specified optionally as part of an objective* $o_i = (on, CA, sa, ca, L_o)$*. It defines the minimal threshold on a relative scale (i. e., 0 to 100 %) that has to be reached for the particular numerical context attribute to fulfil the objective* o_i *(i. e., to evaluate the satisfied condition positively).*

We took this concept from the field of *Approximate Computing* as it promises to reduce costs and overhead introduced by additional computations for analysis of objectives and execution of compensating actions in error cases [HO13]. With

respect to the lighting scenarios, the objective "enough light for working" could already be fulfilled at 90 % of the necessary light levels in the context of the smart home ($L_o = 0.9$) and it has to be at 100 % in the smart factory ($L_o = 1$). With this, we can scale the level of consistency and precision to be reached when ensuring cyber-physical consistency (*Soft Consistency*), probably reducing the number of computations necessary to restore CPC. More abstract non-numerical context attributes are usually compared based on the equality/comparison function for the respective data type (e. g., process states are compared by checking for string equality, which can only be true or false). Depending on the comparison operators used within the satisfied conditions, the consistency level defines a minimal threshold (*larger than*), maximal threshold (*smaller than*) or a range (*equals*) for numerical context values to be reached to fulfil the condition. When relying on ranges or consistency levels, the definitions of CPC_t and $\neg CPC_t$ in Equations 4.1 and 4.2 have to be adjusted to reflect the loosened constraints for equality of the assumed state $S_{C,t}$ and the actual state $S_{P,t}$. The consistency level can also be regarded as a guarantee in the form of a QoS contract for CPS workflows, which ensures that certain QoS, KPI or CPS context levels have to be reached during the execution.

4.7. Consistency Style Sheets

Processes in CPS can become very complex–consisting of various subprocesses and activities that influence multiple virtual and physical context factors. Processes may have several *cyber-physical* process steps or need to fulfil other QoS and KPI criteria in *managed* process steps. Thus, goals associated with process steps may have multiple complex objectives with respect to different consistency requirements, aspects and views, which may bloat the process models. We therefore introduce *Consistency Style Sheets* for workflows–in analogy to *Cascading Style Sheets* containing style-related parameter configurations for websites [BLLJ98]. The Consistency Style Sheets contain the MAPE-K (managed) configurations of an entire process concerning these views–goals, objectives and consistency levels to be used to check the outcome of the process execution for managed process steps and to adapt a process step instance in case of unfulfilled objectives. The goals contained in a style sheet may refer to various views, e. g., regarding criteria related to CPS effects, distribution, QoS levels, process conformance, and other consistency aspects. In general, Consistency Style Sheets contain the parameter configurations for the individual process steps of a process model used in the MAPE-K feedback loops to manage their executions. At runtime, the style sheets are parsed by the respective WfMS or *Feedback Service* and the process parameters are set before an instance of the process is executed (e. g., as described in Figure 6.11 in Section 6.12.1).

Goals are linked to their respective *managed* process steps through the process step's unique identifier specified in the style sheet. Objectives could also be assigned in a more template-oriented way to process steps of a certain type rather than to a specific process step. The sheet is used to configure the corresponding process by setting the relevant attributes (goals and objectives) for its managed elements (process steps) during process deployment. Having this mechanism simplifies the configuration and adjustment of a process and facilitates reuse of style sheets (or style sheet fragments) for other processes in the form of templates. It also sepa-

rates the concerns of specifying the newly introduced concept of goals for processes from the "original" process model and workflow concerns [JGVDSJ14], i. e., models of "legacy" workflows do not have to be altered, only complemented by new Consistency Style Sheets (cf. Section 6.12).

Listing B.1 shows a comprehensive consistency style sheet for an extended version of the *Morning Routine* scenario process containing multiple goals and objectives in JSON. Three goals specify *Cyber-physical* properties of three different process steps: there should be enough light for reading in the kitchen (Lines 3–19), the brewing of the coffee is successful if the temperature of the cup is above 37 degrees (Lines 21–33), and the paper fetching robot is at the correct position for picking up the paper (Lines 35–49). The first goal's objective regarding the change within the illumination values contains a consistency level definition (Line 9) specifying that the light levels are already sufficient at 90 % of the specified objective. The style sheet also contains a goal regarding the *Distribution* of a certain subprocess stating that the battery level of the robot should be above 30 % to guarantee the successful execution of the distributed subprocess (Lines 51–62). The fifth goal (Lines 64–75) specifies criteria for the correct execution of a process step (*Process Conformance*). The step was executed successfully, if it is in state "executed", an error occurred if the execution peer cannot be reached within 5 seconds while still executing an instance of the process step. The process steps from this consistency style sheet will be further described and evaluated in Chapter 7.

The information contained in a consistency style sheet can be used to compute the overall consistency level for a process as a sum of the individual levels and for adjusting objectives with respect to computational costs and levels of precision (*Scalable Consistency*). The process configurations are easily adjustable by changing the style sheets accordingly, which simplifies setup, test and repetitions of experiments when determining optimal process parameters. The conditions defined in a consistency style sheet can be used as a basis for specifying contracts between the virtual world process execution and the physical world process execution [WN14].

4.8. Tools for Modelling of CPS Workflows

After the description of elements and concepts necessary to specify CPS workflows and associated CPS entities, we present the tooling infrastructure to apply the new concepts and model the respective processes for CPS. We developed multiple applications to support users with creating CPS workflows and the corresponding CPS entities–also to provide more sophisticated means for workflow modelling and human interaction (Requirement *R3*). These tools are related to the *Design* (Modelling) phase of CPS and CPS workflows.

4.8.1. Workflow IDE

In accordance with the process modelling language and extensions suggested in the previous sections, we developed a process model editor based on the Graphiti[2] tooling infrastructure framework as Eclipse plugin. As the process metamodel is an

[2] `https://eclipse.org/graphiti/`

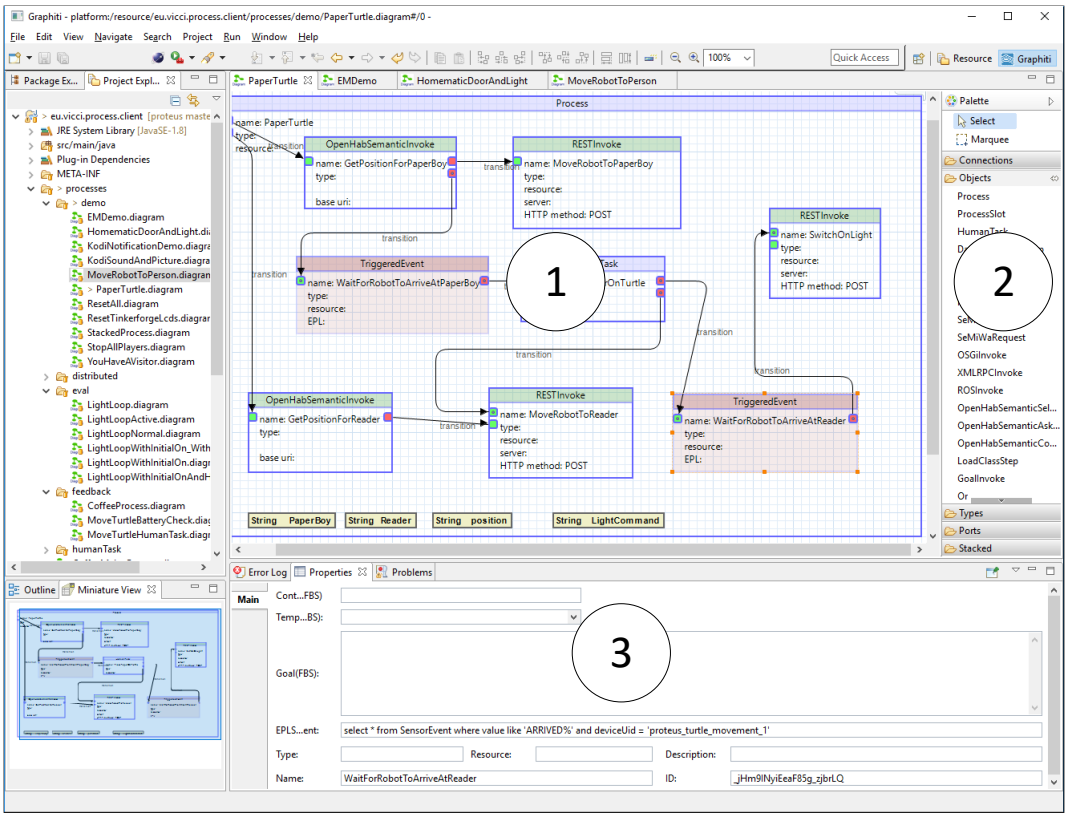


Figure 4.21.: Eclipse-based Process Model Editor.

Ecore model from the Eclipse Modeling Framework (EMF)[3], we decided to use the Graphiti framework for creating the editor. Figure 4.21 shows the Integrated Development Environment (IDE) for CPS workflows as described in [SKNS13] with the canvas for modelling a process (1), the set of graphical process modelling elements (2), and the properties view for defining the attributes of the individual process elements (3). The workflow designer first drags the desired process element (i. e., process step or one of its specializations) from the list of available elements (2) and drops it onto the modelling *Canvas* (1). Then, he/she configures parameters of this process step in the *Properties* view (3). Following, the designer creates ingoing and outgoing control flow and data flow ports–and probably the associated new data types–for the process step from the list of modelling elements. These ports are then connected with the respective ports of other process steps by choosing a transition from the list of modelling elements and selecting source and target ports. The editor also provides some basic means of verification and constraint checking regarding aspects that are not reflected in the metamodel, e. g., the checking of type compatibility between data ports or the checking of mandatory input fields. We also support the use of process fragments and templates from a connected *Process Repository* within the workflow IDE (cf. Section 5.2.7).

With this IDE, the process designer is provided with a tool focused towards end-user development of workflows using drag and drop gestures. However, due to the

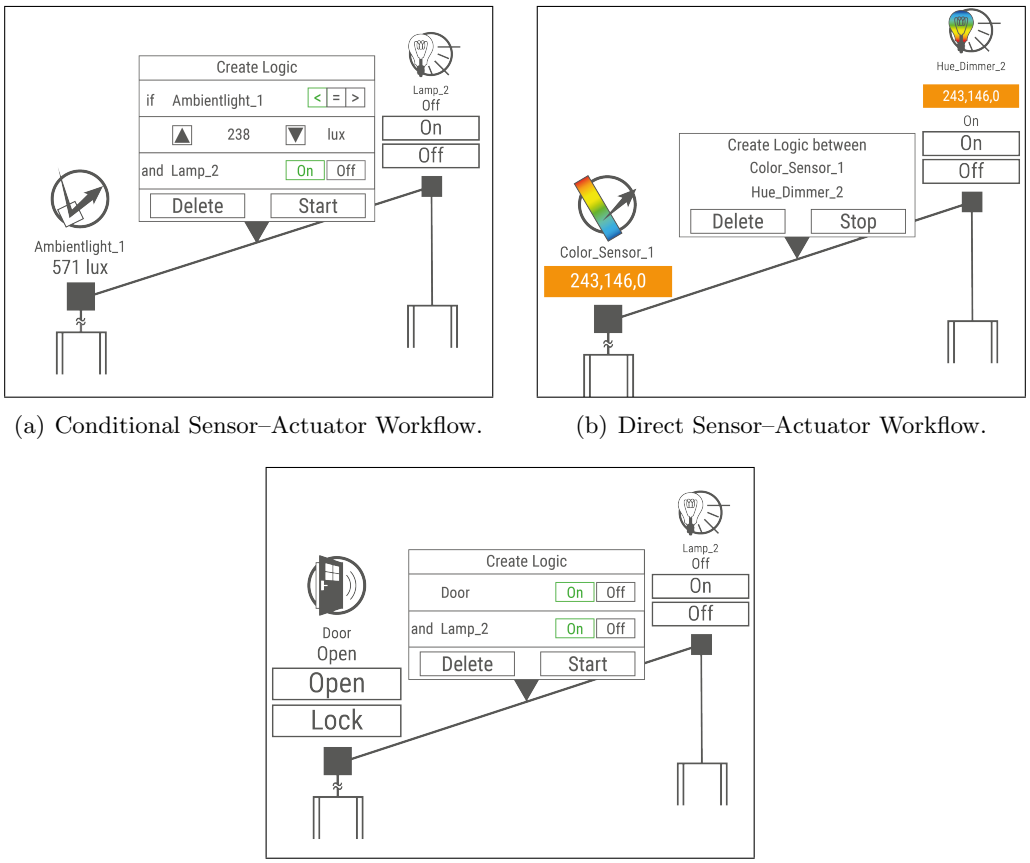
[3] `http://www.eclipse.org/modeling/emf/`

rather technical notation and components of a process, the designer needs deep knowledge of the process composition (e. g., regarding the modelling of data flow between ports), technical details (e. g., regarding specific service parameters), as well as domain knowledge (e. g., regarding the definition of the effects of a process step in objectives). An extension of the IDE to support the use of gestures to draw graphical process elements is described in [NSKS14]. Future developments to increase the usability and end-user friendliness of the workflow IDE could comprise providing a predefined set of available process steps that can be easily composed (wired) by drag and drop from a process repository (cf. Mixed reality workflow composition with *HoloFlows* in Section 4.8.2); DSLs and corresponding extensions of the IDE to ease the definition of goals and objectives, EPL patterns and SPARQL queries with live recommendations and auto-completion using the knowledge base [MVKM16]; as well as a more sophisticated Consistency Style Sheet editor. As the knowledge base is founded on a semantic ontology, suitable ontology editors (e. g., the TopBraid Suite[4] or Protégé[5]) can be used to extend and modify the underlying models. Future development stages may include a workflow store/ecosystem that lets people contribute, share, sell and rate their cyber-physical workflows created with the IDE.

4.8.2. HoloFlows: Mixed Reality Workflow Editor

A first prototype of a mixed reality application for creating basic CPS workflows compatible with the metamodel was also developed as part of addressing the requirement of ubiquitous human interaction (Requirement *R3*) in the context of this thesis with a special focus on creating an end-user friendly intuitive workflow modelling application. The *HoloFlows*[6] app implemented for smart glasses (Microsoft HoloLens[7]) uses mixed reality technology to display holograms containing information about current sensor and actuator states as well as control functionality above the respective physical device (cf. Figure 5.19) [SKGA17]. Besides direct interaction with sensors and actuators, users are able to create simple workflows between actuators, ECA rules between sensors and actuators as well as direct sensor–actuator workflows by connecting the respective devices via a virtual wire in augmented reality. These workflows can be mapped to the workflow metamodel described in Section 4.2 and then executed by the PROtEUS WfMS (cf. Chapter 5). Due to the augmented reality technology displaying sensors, actuators and workflows at their respective physical locations, end-users are able to explore their surroundings and compose simple workflows in a relatively easy way, which enables them to automate, customise and individualise their own CPS processes. The HoloFlows app exploits the physical location of CPS devices as one important context factor of CPS to provide an increased usability and user experience. In its current state, the app is a first prototype for composing CPS workflows compatible with the core classes of the workflow metamodel–supporting the creation of control flows among sensors and actuators as described in the following sections. More advanced concepts (e. g., data flow, complex events, human interactions, distribution, dynamic services or goals for managed process steps) cannot be modelled with HoloFlows, yet.

[4] `https://www.w3.org/2001/sw/wiki/TopBraid`
[5] `http://protege.stanford.edu/`
[6] `https://github.com/IoTUDresden/HoloFlows`
[7] `https://www.microsoft.com/de-de/hololens`



(a) Conditional Sensor–Actuator Workflow.

(b) Direct Sensor–Actuator Workflow.

(c) Actuator–Actuator Workflow.

Figure 4.22.: Mixed Reality App *HoloFlows* for Simple Workflow Composition.

Conditional Sensor–Actuator Workflows: These ECA workflows are created by connecting a sensor with an actuator (cf. Figure 4.22(a)). Upon drawing a virtual connection between these devices by first selecting the particular sensor and then selecting the corresponding actuator (cf. Section 5.6.3), a *condition* defining the threshold for triggering an *event* related to the sensor's value has to be defined. Second, the *action* to be activated by the actuator is selected. The workflow can then be started, it will trigger an event and with that the activation of the actuator once the defined condition related to the sensor value becomes true.

Direct Sensor–Actuator Workflows: These workflows are created by directly connecting sensors producing continuous data with actuators consuming continuous data that is compatible with each other (cf. Figure 4.22(b)). The sensor emits data, which is directly used as input parameter for the actuator once both devices are connected and the workflow is activated. Examples of these type of workflows can be the direct mapping of the color value detected by a color sensor to a connected lamp able to show this color, or the direct mapping of the state of a potentiometer (0 .. 100 %) to the light level of a dimmer switch (0 .. 100 %).

Actuator–Actuator Workflows: These workflow are created by connecting an actuator with another actuator (cf. Figure 4.22(c)). Upon drawing a connection between these devices, the action to be activated for the first actuator has to be selected, and then the action for the second actuator. Once the workflow is started, both actions are executed sequentially in the order they were defined.

4.8.3. CPS Modelling Process

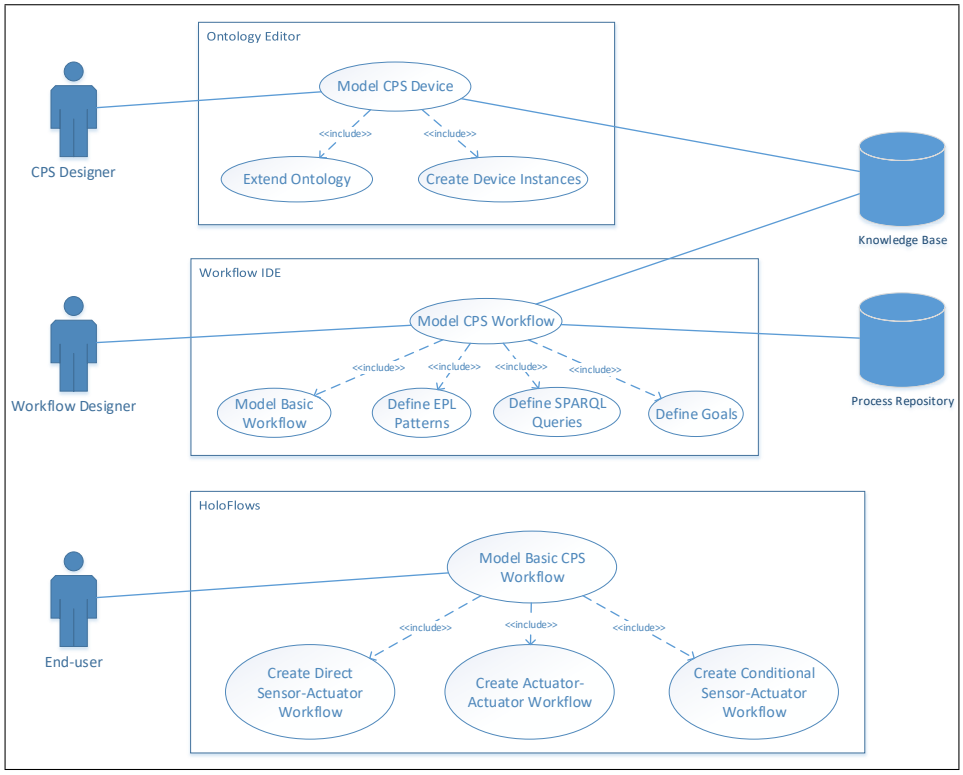


Figure 4.23.: Modelling Activities of Different Types of Users of CPS.

Figure 4.23 shows the modelling activities of different types of users of CPS. The main actors involved in the modelling and design of CPS are the *CPS Designer*, the *CPS Workflow Designer* and the *End-user*.

The *CPS Designer* is an expert in the field of CPS and is responsible for modelling and integrating new CPS devices into the knowledge base and existing infrastructure. If the devices–their properties, functionality, capabilities, context, etc.–can already be modelled with the knowledge base's underlying ontology, the CPS designer only needs to create new instances of the respective devices to add them to the existing CPS infrastructure and knowledge base/triplestore; otherwise the modeller has to extend the concepts of the ontology accordingly to represent the new CPS devices and entities. The CPS Designer uses an ontology editor to model the CPS devices.

The *CPS Workflow Designer* is a domain expert, who models domain-specific CPS workflows based on the workflow metamodel. Besides domain knowledge, the designer needs deep knowledge of the metamodel and the specific processes as well as service parameters and data models. He/she uses the workflow IDE to model

CPS workflows consisting of events, service calls, human tasks, control and data flow, process logic, subprocesses and also possibly process fragments from a repository. Besides these basic processes, the designer also needs to specify EPL patterns, SPARQL queries and goals, which are rather complicated expressions that should be supported by more sophisticated tools also connected to the knowledge base.

The *End-user* wants to create very basic CPS workflows to automate and customise simple routines involving only a small number of CPS devices. He/she uses the HoloFlows app to connect the respective virtual devices with each other in augmented reality (AR) and to set some simple parameters for the three types of connections (cf. Section 4.8.2). The AR app requires only very little knowledge about CPS and workflow composition in general and has a steep learning curve [SKGA17]. In order for the workflow IDE presented in Section 4.8.1 to be usable by end-users, non-computer scientists or non-domain experts, the IDE has to be simplified to a large degree. Service calls, events and other workflow components have to be pre-modelled and available from a repository to use simple drag and drop gestures to compose these workflow elements into complete process models [DR09]. The IDE also has to support the modellers with parameter configurations, automatic suggestions and auto-completion, workflow verification, and probably a wizard guiding the user through the steps of defining executable CPS workflows.

4.9. Compatibility with Existing Business Process Notations

Various works discuss the suitability of existing workflow notations to model business processes [WvdAD+06, Bör12] (cf. Sections 2.3.3 and 3.3). As shown in this chapter with our process metamodel and with the discussions of related work in Chapter 3, additional workflow elements are necessary to cover the special properties of CPS and IoT environments and to fulfil the identified requirements.

The more general concepts regarding the composite structure of processes and logic elements to describe (conditional) splits and merges of the control flow can be found in other workflow languages in similar ways. The detailed description of typed data flow consumed and produced by process steps and services as well as mappings among these data are only partially possible with current business process languages which makes extensions to these notations necessary.

The support of complex rules to define patterns within event streams and model complex high-level events also requires extensions to existing notations [BBDC+15]. Simple sensor-related events can often be modelled with built-in workflow elements related to the occurrence of an event as trigger for following actions or decisions (e. g., event-based gateways in BPMN 2.0).

Many BPM systems support the definition of *Service Tasks* to invoke web Services, usually deployed and registered locally in the WfMS or accessed based on SOAP. The workflow metamodel proposed in this chapter supports these invocations as well, which allows a direct mapping to other BPM notations. In addition, it covers a wide variety of other web service protocols and proprietary services due to CPS consisting of highly heterogeneous resources that often already provide built-in web service interfaces with REST being the prevalent protocol in IoT. The underspecification of workflow activities to enable dynamic discovery of suitable services at runtime is only partially supported by existing workflow notations (e. g., *Worklets*

in YAWL [ATHEVDA06]). New workflow elements are necessary to realise the semantic resource discovery described in Section 4.4.

The concepts of a *Human Task* or similar manual tasks can also be found in existing BPM notations, which allows for a direct mapping to our concepts.

The specification of CPS aspects and other QoS, KPI or context criteria in goals and objectives is not supported by common BPM notations. These aspects have to be specified either as newly introduced metamodel attributes and classes or as new parameters for the particular process steps. The style sheet mechanism proposed in Section 4.7 facilitates the separation of concerns with respect to modelling the "regular" workflow and defining the outcome, effects or success criteria of the process steps in separate documents, which are evaluated during the execution of the MAPE-K feedback loops. Workflows can be defined using the respective notation in its original form and Consistency Style Sheets can be added to a workflow definition. These style sheets may include goals concerning different aspects (e. g., CPS effects, process conformance or distribution) that are used to configure additional process parameters at runtime. The goals are linked to the original workflow definition via the identifiers of the corresponding process steps. This way existing "legacy" process models can be reused without any modifications (cf. Section 4.7).

The workflow metamodel described in Section 4.2 can be viewed as a more technical, imperative workflow language for implementing executable processes, which may be used as an intermediate language between high-level business process notations (e. g, BPMN) and very technical descriptions of processes on the programming level. An evaluation of the expressiveness of our workflow langugage with respect to the *Workflow Patterns* [vDATHKB03] as well as the derivation of transformations or automated mappings to other workflow languages remain tasks for future work.

Relation to BPMN 2.0

BPMN 2.0 is the most widely used process notation to describe business processes in an organizational context (cf. Section 2.3.3). In the context of this thesis, we use BPMN 2.0 to describe example processes from a more abstract perspective to show organisational aspects and to illustrate general concepts and workflows. From our evaluation of existing workflow notations, we conclude that BPMN 2.0 lacks expressiveness and technical detail (e. g., with respect to describing the automated data flow between services, service tasks and other process steps) to implement processes for CPS to be executed completely automatically by a BPMN-based WfMS without requiring human intervention–unless modelled as part of a workflow (*Human Task*). The semantics of some of the modelling elements (e. g., related to events and services as well as activities/tasks) are rather ambiguous and not precise enough to implement fully automated processes for CPS. Partial (possibly automated) mappings between modelling elements of our workflow notation and BPMN 2.0 are possible, e. g., related to simple events, timeouts, service invocations and human activities. To handle errors and other exceptional behaviour during process execution, BPMN 2.0's compensation events and compensation handlers could be used as part of a process-based feedback loop [RS16]. Our proposed process modelling language can be seen as a more concrete technically oriented process notation that BPMN can be used on top of, but that still abstracts from many implementation details of the underlying CPS control applications and processes (cf. Section 2.5.1).

The investigation of related work regarding the modelling of CPS workflows in Section 3.3 shows that BPMN 2.0 lacks most of the modelling elements and concepts necessary to describe CPS workflows with respect to the identified requirements *R1–R8* (e g., complex sensor events, dynamic actuator service selections, or interaction of distributed WfMSes). Many of the related approaches propose extensions of the BPMN language and corresponding implementations or adaptations of the related WfMSes (cf. Chapter 3), which shows that BPMN 2.0 per se is not suitable for describing and enacting the complex interactions of sensors, actuators, smart objects, humans and the physical environments of CPS on the business process level as required within the context of this thesis. These limitations are due to the fact that BPMN was designed for modelling and executing business processes. The extensions and modifications to BPMN 2.0 and a corresponding WfMS necessary to fulfil all requirements would be very complex and invasive, which is why we decided to base our CPS WfMS on an extensible and more implementation-oriented component-based workflow language. Some of our concepts, e. g., goals for defining success and error criteria [KK12], EPL patterns for defining complex sensor events [BBDC+15] and special semantic process tasks for dynamic service selections can be applied to BPMN 2.0 as well. The investigation of the applicability of our CPS workflow concepts to BPMN and an associated WfMS remains subject to future work.

5. Architecture of a WfMS for Distributed CPS Workflows

> "What the gears cannot do the computer might. The computer is the Proteus of machines. Its essence is its universality, its power to simulate."
>
> Seymour Papert

5.1. Introduction

This chapter describes selected aspects regarding the runtime view and necessary components of a workflow management system to implement and execute CPS workflow instances conforming to the CPS workflow notation presented in Chapter 4. New modelling elements were introduced to fulfil the identified requirements for establishing workflows in CPS. Along with these extensions, new software components become necessary to execute instances of these modelled CPS workflows. As pointed out in Sections 3.2 and 3.4, existing workflow engines from industry and research fulfil the requirements only partially, which is why we propose a new system architecture of a CPS workflow management system called *PROtEUS*. This WfMS is meant to serve as a reference architecture for workflow systems in CPS as it discusses rather generic components that can also be added to existing WfMSes.

The WfMS architecture features components for processing complex event streams from various sensors; the dynamic selection and invocation of services according to context, availability and capability constraints of the respective CPS resources; for users to interact with the WfMS using stationary or mobile devices; and to create hierarchical networks of WfMSes that are able to distribute and execute process fragments among each other. These components are integral parts of a WfMS for CPS to fulfil the identified requirements. With the *PROtEUS* WfMS for CPS, we describe the basic concepts and their implementation regarding the composition and interaction of these parts in the form of a possible reference architecture for a WfMS that is able to execute distributed CPS workflows. System engineers can use this architecture as a reference to adapt and extend existing WfMSes with new components according to the principles and concepts described in this chapter.

5.2. PROtEUS Process Execution System

5.2.1. Overview

The *PROtEUS*[1] process execution system is our proposal of a system architecture for a CPS WfMS. In the following, we describe its key architectural concepts that are based on the elaborations in [SKNS15, SHS15, SHS16, SHH17, SNS14b]. We start with explaining basic internal components of the WfMS used to enact processes and interact with sensors and actuators (Requirements *R1* and *R2*). Following, we focus our elaborations on associated external services and applications used in combination with PROtEUS to interact with dynamic services via an IoT middleware (Requirement *R2*), with other PROtEUS instances in distributed processes (Requirement *R4*), and with humans for ubiquitous interaction (Requirement *R3*).

An overview of the architecture and implementation of the *PROtEUS* system–its associated components and used technologies–can be found in Figure 5.1. The focus of our developments is on the engineering aspect of integrating existing and well-established technologies and software components into a comprehensive WfMS for CPS following the general suggestions and WfMS reference architectures presented in [DLRM+13, GdV98, KM07]. Where necessary, we implemented core functionality for executing CPS process instances, additional components and adapters for connecting the system components with each other ourselves. The following sections of this chapter explain the PROtEUS system, its individual software components, associated external services and their interactions in more detail.

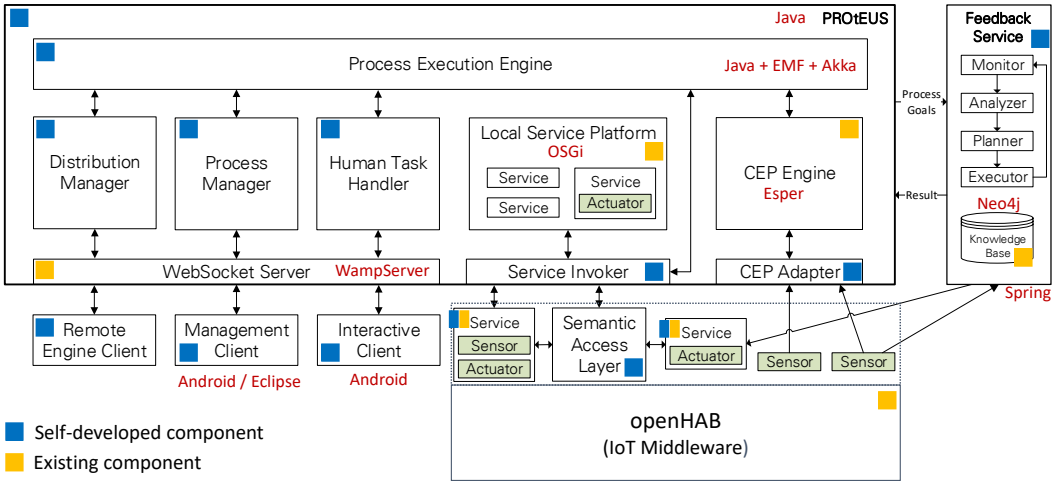


Figure 5.1.: Components and Technologies of the PROtEUS WfMS.

- *Process Execution Engine*: Executes process instances according to their underlying process models (cf. Chapter 4) and communicates with all other components. (Section 5.2.2)
- *Process Manager*: Provides interfaces and functionality to manage, control and monitor the execution of processes. (Sections 5.2.3 and 5.6.3)

[1] `https://github.com/IoTUDresden/proteus`

- *CEP Engine*: Enables complex event processing within multiple CPS sensor event streams to detect specific event patterns defined in the corresponding process steps [SHS16]. (Section 5.2.4)
- *Local Service Platform*: Enables the local deployment and discovery of web services based on OSGi. (Section 5.2.5)
- *Service Invoker*: Supports the invocation of various types of web services to invoke CPS device (sensor and actuator) functionality based on standard or proprietary protocols. (Section 5.2.5)
- *Web Socket Server*: Enables the bi-directional communication and interaction with users and other instances of the WfMS for distributed process execution via publish/subscribe and remote procedure calls. (Section 5.2.6)
- *Human Task Handler*: Distributes manual tasks that are part of a process requiring human interactions [SLSS16]. (Section 5.6)
- *Distribution Manger*: Enables the distributed execution of subprocesses via subcontracting and instance migration on remote *PROtEUS* peers in a hierarchical network managed by super-peers (*D-PROtEUS*) [SNS14b, SHA17]. (Section 5.5)

PROtEUS interacts with the *Semantic Access Layer* (SAL) to dynamically discover and invoke services from an IoT middleware (cf. Section 5.3) interacting with sensors and actuators based on required functionality and context constraints defined in an ontology [HSKS16b, HSK+16] (cf. Sections 4.4 and 5.4). The *Feedback Service* is a generic implementation of the MAPE-K [IBM05] control loop to add self-management to WfMSes and manage the process executions based on goals (cf. Section 4.5 and Chapter 6).

5.2.2. Process Execution Engine

The *Process Execution Engine* creates instances of process models and routes control and data flow through the process instance in accordance with the behaviour of Petri nets. As discussed by Van Der Aalst in [VDA96], Petri nets are a suitable foundation for workflow systems due to their formal semantics and analysis techniques (cf. Section 2.3.3). Despite formal verification being out of scope of our work, we base the execution behaviour of the process engine on Petri nets to allow for a formal verification of the software using processes to control the CPS in future work [BG11]. The verification of the *physical* aspects and effects of processes based on Petri nets requires further research. Our contributions regarding *Cyber-physical Consistency* may be a first step towards this direction (cf. Sections 4.6 and 6.3).

Runtime Behaviour

The engine manages the lifecycles of process steps and their associated ports along with the data instances during execution. At runtime, ports have an activation state, which is used to decide on the point of execution of the according process step instance (Level $M0$ in Figure 4.1). When instantiating a process model (Level $M1$

in Figure 4.1), all of its ingoing control and data ports have to be instantiated and activated with required data instances, respectively. Upon activation of a port instance, its connected transition and target port at the target process step are activated as well. In the regular case, a process step instance is executed once all of its ingoing ports are activated. Special process steps for steering the control flow (e. g., XOR or OR elements) show a different behaviour in accordance with their logical function, e. g., OR activates its outgoing ports if one of its ingoing ports is activated, which disables the rest of its ingoing ports. The engine calls the *execution* function of the current process step and the logic will be determined and executed according to the type of process step using polymorphism. The various types of process steps are described in Section 4.2. Their actual execution behaviour is described in the following sections.

Ports can have an *Optional* attribute, which indicates that the particular port does not necessarily have to be activated to execute the respective process step. The process step's outgoing data and control ports are activated after the successful execution of the process step, which also triggers the activation of connected transitions and target ports. Connected data ports are activated with instances of the data from the corresponding source data ports or according to the data mappings defined for the process steps (cf. Section 4.2.6).

To draw a simplified analogy with the execution behaviour of a Petri net [Pet81], one could state that *Places* describe the current state of a process step instance and its ingoing ports; *Tokens* are control flow or data flow activations that move through the process instance along process transitions (Petri net *Arcs*). In the regular case, a Petri net transition is fired (i. e., a process step instance is changing its state to *executing*), once all its input places consumed a token (i. e., all of the process step's ingoing non-optional data ports and control ports have been activated). After the firing of the transition (i. e., successful execution of the respective process step instance), tokens are created for the output places (i. e., outgoing data and control ports) of the process step instance, which then move along the next arcs (process transitions) to the next process step's ingoing ports (input places). As the tokens moving through a process also have to represent typed data and processes can be composed hierarchically, a *coloured* and *hierarchical* Petri net could be suitable to formally represent the execution behaviour [HJS89]. The detailed investigation of this mapping including the concepts of (sub)process steps, ports, transitions, activations and the extensions described in the following sections as well as the implementation with the help of this kind of Petri nets remains subject to future work.

Runtime Attributes

Upon instantiation, process steps, ports and transitions have additional attributes that are used for managing and executing processes. The lifecycle of a process step instance is depicted in the following state chart (cf. Figure 5.2). Upon uploading a process model file created with the IDE to the WfMS, the model (Meta-level *M1*, cf. Section 4.2.1) and all its process steps are in state *undeployed*. Calling the process manager's *Deploy* function causes the process model to be loaded into memory and some general process parameters to bet set. From that point on, process instances (Meta-level *M0*, cf. Section 4.2.1) of the process model can be created. Multiple pro-

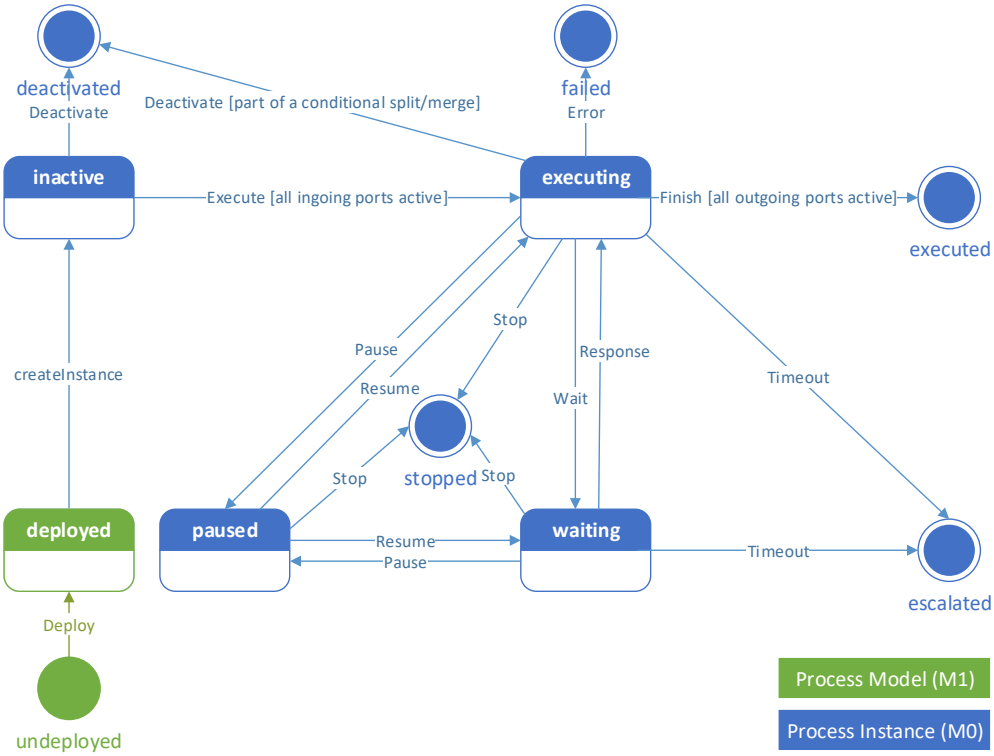


Figure 5.2.: Lifecycle of a Process Step Instance.

cess instances can be created from a process model. During instantiation (transition from level *M1* to *M0*), new objects are instantiated for the individual process steps, transitions, ports and data–all containing unique identifiers, references to the corresponding process model elements, and individual states (*inactive*). These elements also contain a reference to the particular process instance they are part of, which is also in the *inactive* state. These states and additional runtime attributes of the process instances and their components are described by a dedicated *Runtime Metamodel*, which also references the process metamodel. We introduce this dedicated metamodel to decouple the design time metamodel used for modelling processes (Level *M1*) from the runtime metamodel used for describing process instances at execution time (Level *M0*).

The invocation of the *Execute* method triggers the process step's and containing components' states to be switched to *executing* under the condition that all non-optional ingoing ports of the process step instance are active. Special process steps for directing the control flow (e. g., XOR and OR) may show different behaviour at this point with respect to their logical function. A process step instance can also become *deactivated* in case it is part of a process branch that is not executed due to a conditional split or a disjunctive merge (e. g., if an OR process step is activated by one of its ingoing branches, the process step instances within the other branches are deactivated). Loops show slightly different behaviour, too. Once a loop iteration is executed, the process steps contained in the loop are instantiated anew.

From the state of *executing*, a process instance can be *paused* and resumed or *stopped*, which terminates its execution. In case all non-optional ports are activated

and the process step is executed successfully, the instance is *executed*. The occurrence of an error leads to the transition to the *failed* state–except for the case that the MAPE-K feedback loop is able to compensate the error (cf. Chapter 6). In case of asynchronous operations, e. g., when executing a human task or a distributed process, the process instance enters the *waiting* state, which can also be paused and resumed, as well as stopped. If an *Escalation Port* is part of the process instance (cf. Section 4.2.7) and the specified timeout is triggered, the process instance moves from the states of *executing* or *waiting* to *escalated*.

Instances of transitions and ports are also characterized by their activation states (*active*/*inactive*). Data ports include the concrete data instances as results of the data flow within the process as additional runtime attributes. In case an execution *Resource* is selected dynamically (cf. Section 5.4), this attribute is updated with the concrete resource. The implementation of the lifecycle managers for all process components that are part of the Process Execution Engine relies on the *Akka*[2] framework for building concurrent and distributed applications.

As a future extension of the lifecycle depicted in Figure 5.2 the introduction of an *Enable* transition has to be investigated in analogy with the similar concept for BPMN. In an analysis phase preceding the execution of a particular process step instance, the involved resources should be evaluated with respect to their availability (e. g., reachability of services or event sources) and the process model in general should be analysed to discover deadlocks and unreachable steps. That way, the process execution system is able to ensure that a particular process step or transition **can** be executed and the necessary resources are actually available for static service invocations. For dynamic service invocations (cf. Section 5.4) and process executions monitored by the MAPE-K loop (cf. Chapter 6) this is only partially necessary.

5.2.3. Process Manager

The *Process Manager* component is the interface to manage process models and the execution of process instances. It provides means for uploading process models as well as the parametrisation, deployment and instantiation of process models in the execution system. Process instances can be controlled via the Process Manager and monitoring/logging information about process instances and the execution system in general can be retrieved by clients via the Process Manager. Upon state changes in process instances, the process manager broadcasts a logging message containing relevant data (i. e., process step information, states and data instances) via the WebSocket server. In Section 5.6, we describe the means of interacting with the process execution system via the Process Manager in more detail.

5.2.4. Complex Event Processing Engine

The integration of numerous event sources and with that, the necessity for processing of complex events is identified as one of the main requirements for CPS workflows in Section 2.6 (Requirement *R1*: Complex Sensors). The corresponding process metamodel element *TriggeredEvent* for representing complex events is introduced in Section 4.2.9. The *Complex Event Processing Engine Esper*[3] serves as the event

[2] `https://akka.io/`
[3] `http://www.espertech.com/esper/`

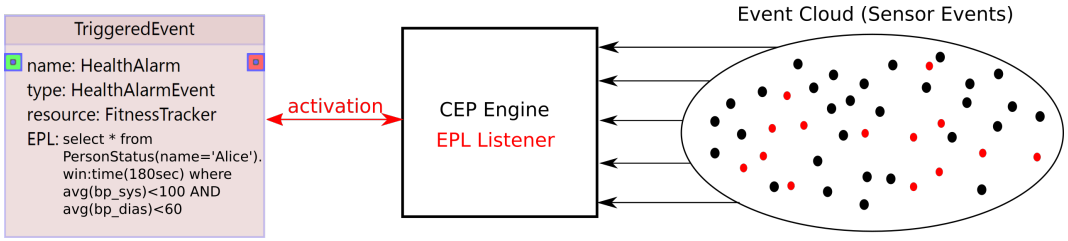


Figure 5.3.: Complex Event Pattern Detection.

processing component in PROtEUS. Its communication with other components and the relation of events to the process metaclasses are shown in Figure 5.4 by means of an exemplary process modelled with our process notation. The exemplary event *HealthAlarm* is an instance of the *TriggeredEvent* metaclass, which adds an EPL statement to the particular process step (cf. Section 4.2.9). Figure 5.3 shows the flow of events and pattern-based event processing within our approach.

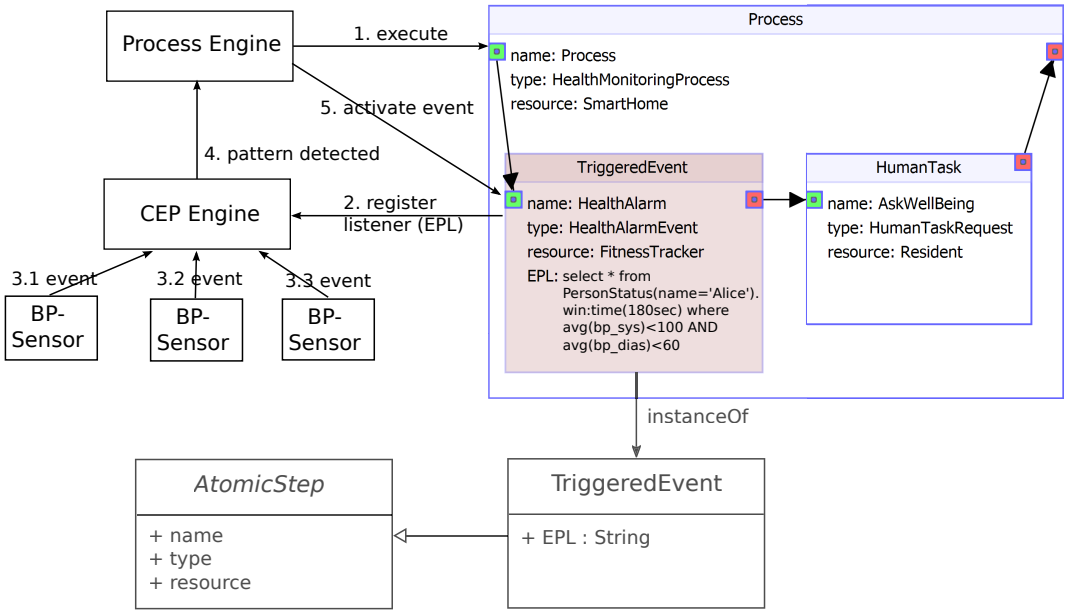


Figure 5.4.: Communication between Process Engine and CEP Engine During Process Instance Execution.

Upon the start of the PROtEUS runtime environment, the CEP engines subscribes and listens to a configurable set of events–the openHAB IoT middleware being the main event source. These low-level event sources are connected to the CEP engine via custom adapters, which provide the corresponding network interfaces and map the events to the CEP engine's internal data model. When reaching an instance of a *TriggeredEvent* process step defined in a process model, the process engine calls the execution method of this type of process step (1) as shown in Figure 5.4. This leads to the registration of a listener for the EPL pattern defined in the TriggeredEvent process step (2). From that point on, the CEP engine analyses the incoming event stream with respect to the EPL pattern. The example in Figure 5.4 shows the anal-

ysis of the blood pressure values of a certain person (*Alice*). A blood pressure sensor sends data about the wearer's health in a constant time interval (3.1–3.3) to the CEP engine. In case the average systolic and diastolic values fall below the defined thresholds (cf. EPL pattern in Figure 5.4) over 180 seconds of time, the pattern is detected by the CEP engine (4), and the process level event (*HealthAlarm*) is activated (5) leading to the continuation of the process instance. In our example, the resident will receive an *AskWellBeing* human task notification on her end-user device enquiring the health status of the resident. New sources and types of events can be added to the CEP engine at runtime via sensor specific adapters/wrappers. Events may comprise low-level sensor events but also higher level and complex events emitted from other processes and process instances as *TriggeringEvents* (cf. Section 4.2.9) as well as from additional software components that aggregate, analyse and produce high level events from low-level data (e. g., activity recognition systems or machine data analysers). The EPL pattern referring to sensors and events of arbitrary granularity has to be defined by the process modeller who has to have knowledge about existing event types and event sources contained in the *Knowledge Base* (cf. Section 4.3).

5.2.5. Service Invoker and Local Service Platform

Services belong to the basic building blocks of distributed systems (cf. Section 2.3.5). When interacting with external components, sensors, actuators and more complex CPS devices, we assume that they provide a service-based interface to enable the invocation of their functionality. Sensors and actuators may be encapsulated by a web service running on the associated embedded device. The *Service Invoker* is the component within PROtEUS, which performs the service calls to the specific service endpoints. Section 4.2.8 discusses specific metamodel classes as specialisations of the *ServiceInvoke* process step related to the type of service being invoked. According to this type, a client performing the service call is instantiated by the Service Invoker with the respective service parameters and input data (*Static Service Invocation* of known resources). Figure 5.5 shows the communication among the PROtEUS components and the process instance to invoke an external REST service. The process engine executes the corresponding *RESTInvoke* process step (1) to place an emergency call in our demo scenario (cf. Section 2.2.2). This leads to the Service Invoker instantiating a REST client (2) and calling the external server hosting the specified REST service (3). The service's response is reported back to the service invoker (4) and returned data is integrated into the process step (5) (cf. Section 4.2.6).

Figure 5.5 also shows the selection and invocation process for dynamic services via the Semantic Access Layer (SAL) component responsible for finding services at runtime (*Dynamic Service Invocation* of unknown resources, cf. Section 5.4). The *DoorUnlocking* step is a specialisation of the *SemanticInvoke* process step for the openHAB middleware (cf. Section 4.4), which relies on a semantic query to be sent to and evaluated by the SAL. When executing this step (6), the Service Invoker sends a request containing the semantic query to the SAL to look for devices and services in the knowledge base that match the criteria defined in the query (7). A more detailed description of this discovery process can be found in Section 5.4. Once the SAL successfully determined a suitable service, it reports its URI and required parameters back to the Service Invoker (8), which will then call the respective service (9). The

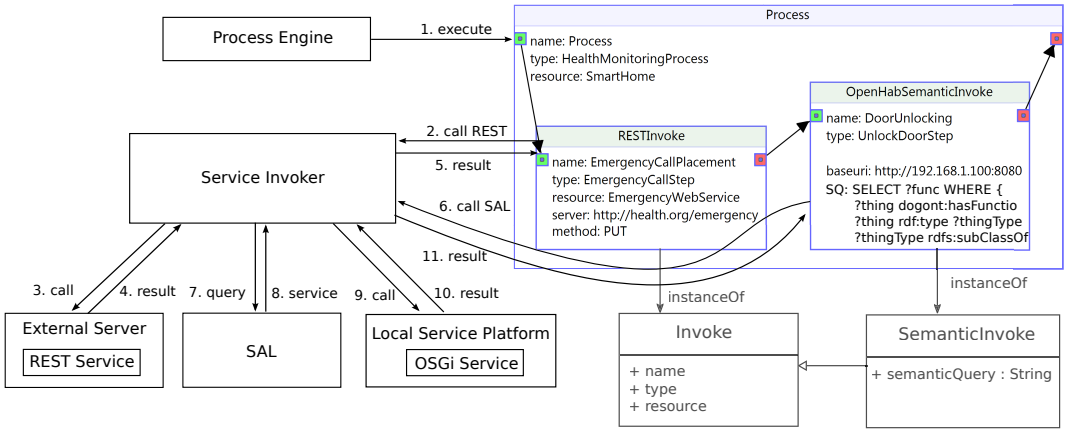


Figure 5.5.: Message Flow during the Execution of Static Service Invocations and Dynamic Service Invocations via the SAL.

results of the service invocation are reported back to the Service Invoker (10) and incorporated into the process (11).

The excerpt from the scenario process depicted in Figure 5.5 shows the use of the *Local Service Platform*, which is also part of the PROtEUS architecture. As there is often a need for deploying local services only accessing in-house devices and applications, e. g., for security reasons–only allowing access to the door opener from the local network, we provide means for installing and running services locally with this component. The OSGi-based platform enables the deployment of bundles and services at runtime. It also integrates a service registry for dynamic service discovery. In our use case, the SAL uses the semantic query to find the service to unlock the door deployed on this local platform (cf. Section 5.4).

5.2.6. WebSocket Server

The *WebSocket Server* component enables asynchronous, bi-directional communication with the PROtEUS environment. It relies on the Web Application Messaging Protocol (WAMP)[4], which provides means for publish/subscribe communication and remote procedure calls. These functionalities are used for asynchronous tasks such as the management and monitoring/logging of process instances via the Process Manager (cf. Sections 5.6 and 5.2.3), the interaction via the Human Task Handler (cf. Section 5.6.1), and the remote execution of subprocesses in a distributed process execution setup via the Distribution Manager (cf. Section 5.5).

5.2.7. Process Repository

The Process Repository is an external component used for persisting process models and parts of process models. It can be accessed by the workflow IDE to store and retrieve process steps to be used in new process models. The process manager also uses this repository to download and instantiate existing processes. Communication with the repository is done via the WebSocket's Remote Procedure Call (RPC)

[4] http://wamp-proto.org/

functionality. The repository uses the Teneo[5] database framework combined with Hibernate[6] for persisting EMF models.

5.3. Internet of Things Middleware

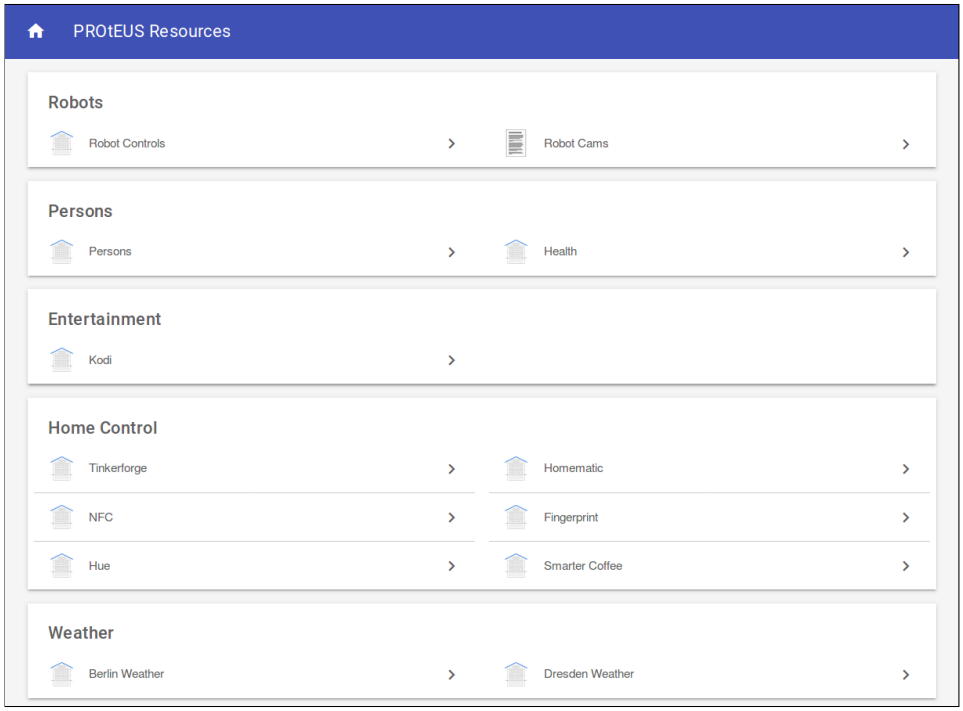


Figure 5.6.: Management Application for PROtEUS Resources in IoT Middleware.

The existing OSGi-based openHAB 2.0[7] platform serves as the central IoT middleware for our smart home setups. Sensors and actuators as well as more complex entities such as service robots are connected to the middleware using already existing bindings or bindings that we implemented ourselves. These heterogeneous CPS resources are managed and unified by the middleware, which offers a RESTful API to access sensor and actuator functionality as IoT services (cf. Section 2.4.4). The middleware provides a web-based user interface to manage available resources (cf. Figure 5.6). We complemented this interface by a mobile Android-based dashboard application for tablet devices as shown in Figure 5.11 and detailed in [SLSS16]. Besides locations and current states of sensors and actuators, the mobile app also allows access to actuators' functionalities and sensor data histories. The middleware is augmented by the SAL (cf. Section 5.4) as new plugin to enable the dynamic discovery and selection of suitable resources and services at runtime [HSKS16b]. This includes an ontology describing the IoT devices' properties, functionalities, capabilities, contexts and relations among each other in the knowledge base (cf. Section 4.3).

[5] `https://wiki.eclipse.org/Teneo`
[6] `http://hibernate.org/`
[7] `http://www.openhab.org/`

5.4. Dynamic Service Selection via Semantic Access Layer

The requirement *R2* regarding the dynamic selection of CPS resources–sensors and actuators–was identified as an important aspect for CPS workflows in Section 2.6. The metamodel extensions enabling the specification of semantic queries to find available instances of these resources at runtime based on information contained in the knowledge base were introduced in Section 4.4. The SAL acts as an intermediate layer between the PROtEUS WfMS and the openHAB IoT middleware to facilitate the dynamic discovery and selection of services as described in [HSKS16b, HSK+16]. The SAL[8] is a new plugin component added to the base version of the openHAB middleware, which can be accessed as a REST service.

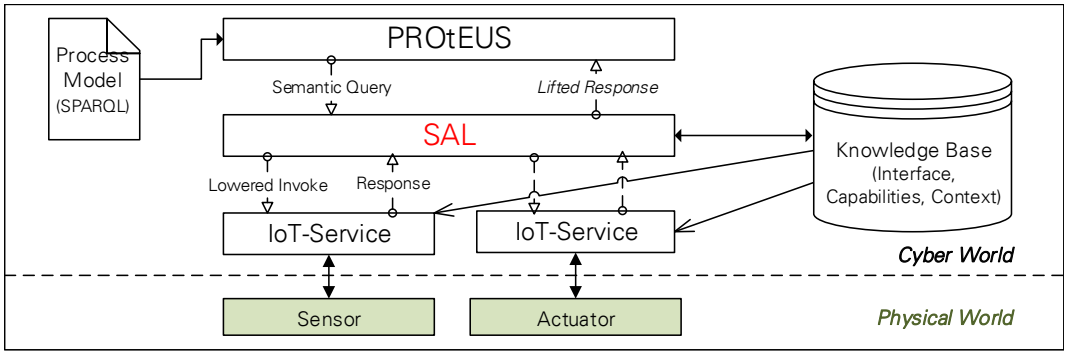


Figure 5.7.: Semantic Access Layer (SAL) as Mediator between WfMS and IoT Services.

Figure 5.7 shows the SAL as mediator between PROtEUS and actuators and sensors accessed via their IoT service interfaces. As described in Section 4.4, the process designer is able to specify SPARQL queries for special types of semantic process steps. These queries contain context and capability constraints as criteria that have to be fulfilled by suitable process resources to execute the particular process step. We support the retrieval of specific sensor and actuator states (*SemanticSelect*), the evaluation of sensor data (*SemanticAsk*) and the invocation of actuator functionality based on the actuator's associated IoT services (*SemanticCommand*) (cf. Section 4.4). Upon execution of a semantic process step, PROtEUS's Service Invoker sends the semantic query contained in the process definition to the SAL. The SAL then evaluates the query with respect to the (instance) information available in the knowledge base. Depending on the type of the semantic process step, the SAL initiates the invocation of the found IoT services to retrieve and check data or to execute a command for sensors or actuators found after evaluation of the query (*Lowered Invoke*). Besides a domain-specific model of their functionalities, capabilities and other context factors, the knowledge base also contains the URIs of the resources' service interfaces to be used by the Service Invoker. After the service is executed, the response of the particular service call is lifted to the process level so that this data can be used in consecutive process steps.

With the SAL, we support the underspecification of process resources for specific tasks. Due to the varying availability of devices in CPS and IoT, the assignment

[8] `https://github.com/IoTUDresden/openhab-distro`

of process resources to a workflow task based on its static URI at design time is not always feasible. The criteria defined in a semantic query are on the more abstract levels of functionalities and capabilities of resources constrained by additional context factors, which allows for the dynamic and context-sensitive discovery and invocation of services associated with the respective IoT devices. More detailed elaborations on applying goals for dynamic service discovery and workflow adaptations for role-based resources in the IoT can be found in [Hub18]. Despite having an ontological foundation and semantic description of the CPS entities and their relations, we do not exploit the full potential of this semantic information, yet. Currently, the knowledge base serves as sophisticated data model and data (instance) storage that can be queried to retrieve instances, parameters and related information as part of semantic searches. The application of semantic reasoning techniques to derive new knowledge from the existing models and instances may reduce the modelling effort and increase the flexibility of the dynamic service discovery for IoT services even further [DBBM11]. A detailed investigation of applying these inference mechanisms for dynamic process resource discovery and selection remains subject to future work.

5.5. Process Distribution

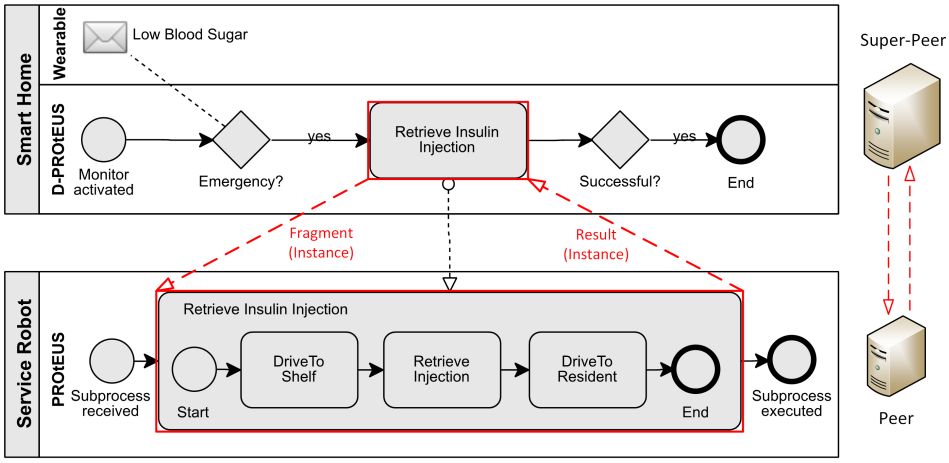


Figure 5.8.: Subcontracting as Means for Process Distribution among Peers and Super-Peers.

The capability of distributed process execution was identified as one important requirement for CPS workflow engines in Section 2.6 (Requirement *R4*). CPS and IoT environments consist of a large number of devices and entities, which are often organized in hierarchical structures. The current trend of *Fog Computing* [SW14] tries to leverage these properties of IoT systems to provide scalability, security and fast preprocessing [BMZA12] by also using processing resources of devices along the network (cf. Section 2.3.5). Network gateways provide computing resources closer to the edge that can also be used for process execution and distribution of tasks. For that reason, process management systems have to be scalable in terms of manageable process resources and the support of decentralized SoS architectures. The PROtEUS WfMS supports these requirements as described in detail in [SNS14b, SHA17].

Figure 5.8 shows an extended workflow based on the Emergency scenario process to illustrate the advantages of distributed process execution in the context of a mobile health care/AAL scenario [PRS+13, PRBA15]. After the detection of an emergency situation by a wearable blood glucose monitor, a service robot is instructed with a subprocess to retrieve an insulin injection from a particular shelf within the smart home. This subprocess consists of three process steps to be performed by the robot: driving to the shelf, retrieving the injection, and driving to the human in need. These three special process steps are executed locally by the robot as an edge device (cf. Section 2.3.5) without any other process resources involved and without the need of communicating with other process engines, which is why the robot is able to enact this subprocess autonomously. The overall process terminates upon the successful execution of this subprocess.

5.5.1. Distributed Systems Architecture

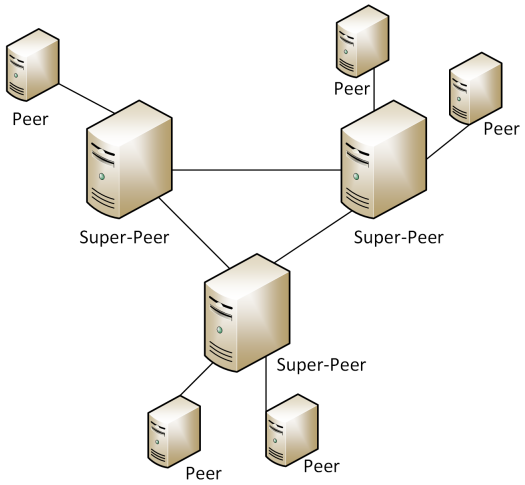


Figure 5.9.: Hierarchical Overlay Network Structure of Peers and Super-Peers.

As suggested in Figures 5.9 and 5.8, we rely on a hybrid overlay network structure, which consists of "regular" PROtEUS WfMS peers and higher order *D-PROtEUS* super-peers [SNS14b]. This structure reduces bottlenecks and single points of failures compared to classical client server architectures and it is also more efficient regarding communication and organisational overhead compared to pure peer-to-peer approaches [YGM01] or fully decentralized systems (e. g., Multi-agent Systems [Fer99]). *Peers* and *Super-Peers* form a multi-level hierarchy providing a scalable solution with respect to the number of process engines and process instances and leveraging locality in the sense of Fog Computing as certain workflow tasks can be moved closer to the computing edges onto specialized devices, which also increases autonomy of these devices (cf. Section 2.3.5).

Peers

Within the hybrid peer–super-peer infrastructure, peers are arbitrary devices that are able to run the PROtEUS system and therefore execute process instances as

described so far. Each peer is connected (statically) to a dedicated super-peer, which manages it and from which the peer receives process fragments and execution commands. Upon the initial connection to the corresponding super-peer, a peer sends its profile including its name, type, IP address, and capabilities to the super-peer. In CPS, peers may comprise various classes of devices ranging from regular desktop computers to resource-constraint embedded and mobile devices.

Super Peers

Besides also running the PROtEUS system for executing process instances, super-peers have an active *Distribution Manager* component, which is responsible for managing the network infrastructure and the distributed process execution. This configuration is called *D-PROtEUS*. The role of a super-peer is assigned manually by the system architects based on a device's available computing resources. Super-peers should be more reliable with respect to availability and have a steadier network connection as well as more computing resources as they have to perform additional management tasks. Due to the super-peers also running the WfMS, they can execute processes and act as "regular" peers to higher level super-peers. In order to maintain a global state of the process executions, super-peers exchange global synchronization information about the execution of process instances among each other. A multi-level hierarchy consisting of peers, super-peers and super-peers, which also act as peers allows for delegating subprocesses and tasks along the paths of the tree-like network hierarchy. Suitable resources for the distributed process execution can be found in the peer–super-peer network in various ways, e. g., by using the *Kademlia* algorithm based on distributed hash tables [MM02].

5.5.2. Distribution Manager

The *Distribution Manager* is the component running on a super-peer, which is responsible for communicating with *Remote (PROtEUS) Engine Clients* via the WebSocket server (cf. Figure 5.1). Its main task is to extract, distribute and merge subprocesses and process fragments to and from peers in the sense of *subcontracting* for processes [vdA00] and process *instance migration* [ZHKL10]. Figure 5.8 shows the subcontracting and migration mechanisms with respect to the "Retrieve Insulin Injection" subprocess instance. The super-peer instantiates the particular process and enacts the process instance using the D-PROtEUS system (0–1) (cf. Figure 5.10). It evaluates the corresponding *Resource* and *Distributed* attributes of process steps and subprocesses (cf. Section 4.2.5). If a subprocess contains the active *distributed* flag and a resource identifier belonging to one of its associated peers, the Distribution Manager is invoked with the corresponding process step as input parameter (2). The Distribution Manager creates a snapshot of the subprocess instance as proposed in [MCS16] and then calls the WebSocket server to serialize relevant data as a process fragment–the process step model, the current state of the process instance and ingoing data instances–and to transfer it to the peer (here: Service Robot running the PROtEUS system) in question (3) (cf. *instance migration* [ZHKL10]). Here the WebSocket server deserializes the received data (4) and invokes the Process Manager to instantiate and execute the subprocess based on the model information, data instances and state of the process (5). The peer's process manager instantiates the

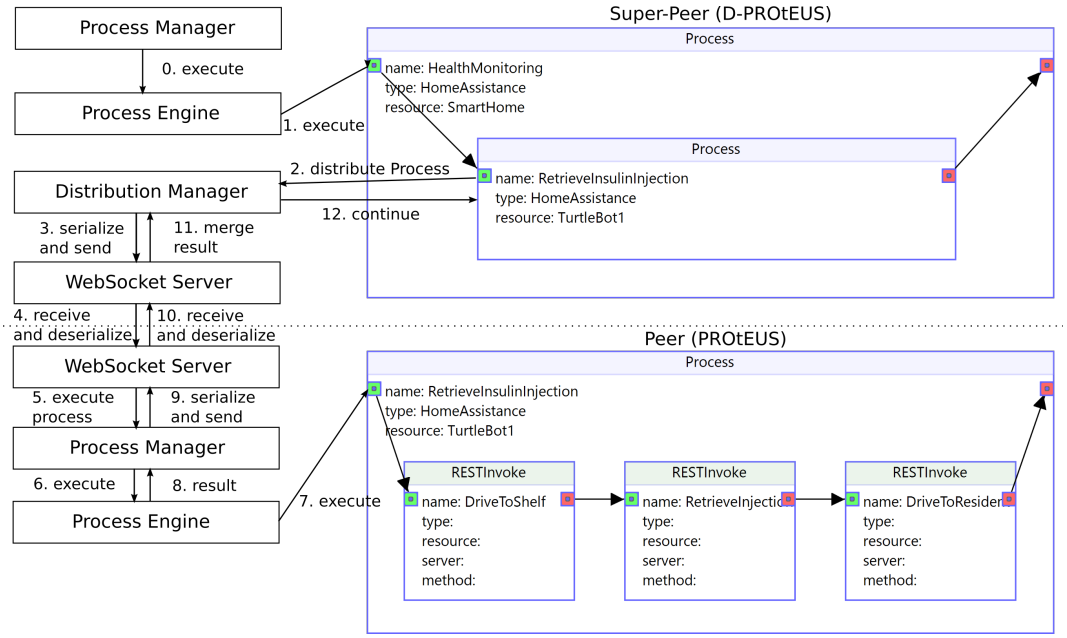


Figure 5.10.: Message Flow during Distributed Process Execution.

subprocess, the PROtEUS WfMS executes the instance (6–7), and the result of the execution (i. e., new states of the process instance and outgoing data instances) is transferred back via the WebSocket server to the super-peer's Distribution Manager (7–10). The Distribution Manager then merges these results into the main process instance (11) and the execution continues on the super-peer (12).

With subcontracting, the hierarchical structure of the network can be leveraged when distributing tasks and processes. Specific process activities and subprocesses can be moved closer to the compute edge for autonomous and fast local preprocessing or to the Cloud for resource-intensive computations (cf. Section 2.3.5). Offline execution is enabled through subcontracting as devices do not necessarily have to rely on a permanent connection to their designated super-peer when executing subprocesses. This increases the resilience of the process execution, especially on mobile devices (e. g., service robots), which are likely to disconnect from and reconnect to the network more frequently. Subcontracting also increases data security as only data relevant for the execution of the specific subprocess is transferred to the peer, while global process data is only known to the responsible super-peers. The selection of suitable peers is currently done in a static way based on the URI attribute contained in the process step model, which points to the specific peer. More sophisticated mechanisms of dynamically assigning resources automatically to subprocesses exist (e. g., as described in [SYY07, MVSA16]), but are out of scope of this work. As the *PROtEUS* system is also available as a Docker container[9], multiple instances of the peer (*PROtEUS*) or super-peer (*D-PROtEUS*) configurations can be started easily and be automatically distributed to various nodes based on different criteria (e. g., by using the Kubernetes[10] container orchestrator). The subcontracting for

[9] `https://www.docker.com/`
[10] `https://kubernetes.io/`

processes as applied in our approach provides a mechanism for the top-down distribution and delegation of tasks in the network hierarchy from higher order peers to lower order peers. This is in contrast to the "classical" way of programming for Fog computing, where processing is done on the nodes closer to the edge first and then data and tasks are moved up the hierarchy (cf. Section 2.3.5).

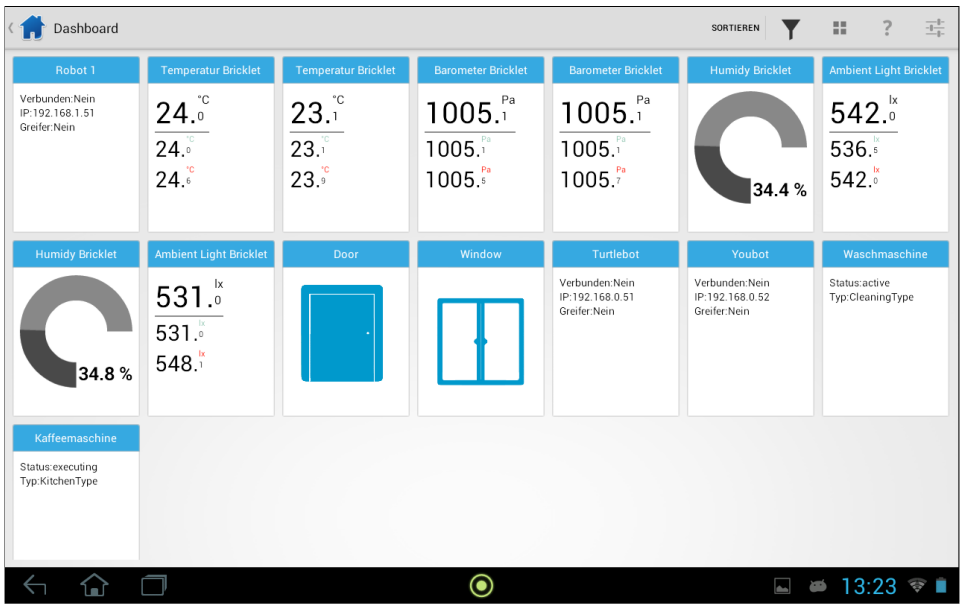


Figure 5.11.: Mobile Dashboard Application for Managing PROtEUS Resources.

5.6. Ubiquitous Human Interaction

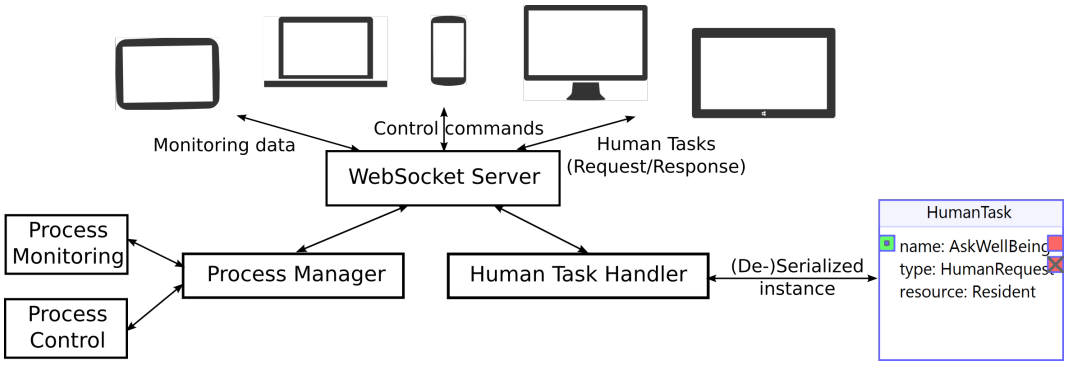


Figure 5.12.: Interacting with PROtEUS via Different Devices and Modalities.

Despite the main goal of an increased automation in CPS via workflows, users are still important factors and stakeholders in CPS as discussed with respect to requirement *R3* in Section 2.6. Therefore, the users have to be provided with various means of interacting with the CPS workflows–either to control and monitor them or to perform *Human Tasks*, which are part of a workflow specification. We have

developed several applications to facilitate the interaction with the PROtEUS system using mobile and stationary devices in the sense of ubiquitous computing [Wei91]. Figure 5.12 presents an overview of interaction means using the WebSocket server in combination with the Process Manager and the Human Task Handler.

5.6.1. Human Task Handler

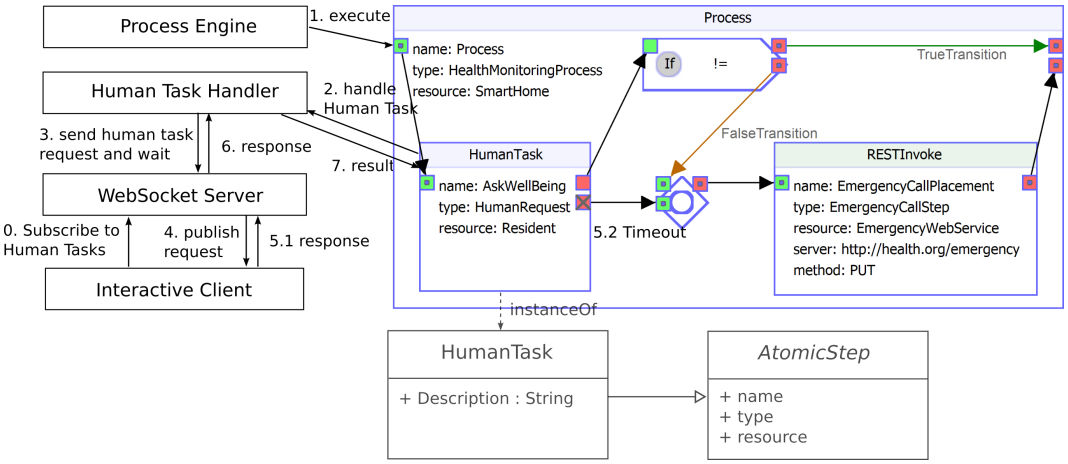
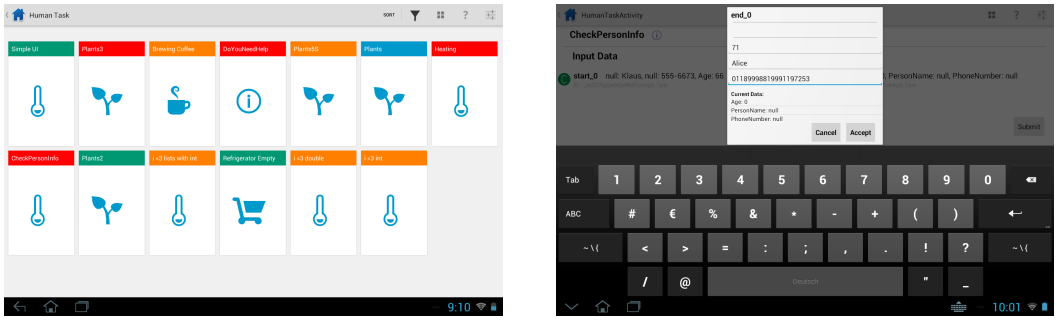


Figure 5.13.: Message Flow during Execution of a Human Task.

The *Human Task Handler* component is responsible for publishing requests concerning user interactions via human tasks as modelled in the respective processes (cf. Section 4.2.10). Figure 5.13 shows the message flow during the execution of a human task instance. Before the execution of a human task (1), Interactive Clients need to be connected to the WebSocket Server and subscribed to the topics that human tasks are published to (0). The Human Task Handler is invoked with the data of the respective process step, i. e., general information–type, name, description–and ingoing data instances as well as the required outgoing data ports (2). It then sends the human task request asynchronously to the WebSocket Server and waits for an answer (3). The WebSocket server publishes this request, which contains the relevant process step instance information (4). In our example, the human task asks the resident of the smart home for his/her well-being. This question is contained in the description of the human task process step. There is no ingoing data necessary, the outgoing data port contains a Boolean value representing the user's answer. In Figure 5.13, the user either sends a response via his/her end-user device (5.1), which will be sent back to the Human Task Handler (6) and incorporated into the *AskWellBeing* process step (7), or there will be a timeout triggered via an *Escalation Port* (cf. Section 4.2.7) after a defined timeframe without receiving an answer (5.2). In case the answer is evaluated by the *If* process step as *false* or the timeout occurred, an emergency call will be placed using a RESTful service invocation. Otherwise the process will terminate.



(a) Human Tasks Overview.

(b) Human Task Interaction.

Figure 5.14.: Mobile Human Tasks Management App.

5.6.2. Human Task Manager

Human Tasks are introduced into the workflow metamodel as dedicated tasks within a process performed by users, e. g., to solve manual tasks, to enter data or to handle errors that have occurred (cf. Section 4.2.10). The Human Task Handler is called by the process engine once it reaches an instance of a human task process step. It serializes and publishes this process step instance via the WebSocket server on a specific topic for human task requests. Interactive devices subscribed to that specific topic receive these requests and present a corresponding interface for handling open human tasks to the user as shown in Figures 5.14(a) and 5.14(b) for an Android-based human task management app. The app's overview section serves as worklist handler showing a list of all work items (human tasks) for the particular user. The human task instance data contained in a request is used to create a simple user interface automatically using basic Android user interface elements. The task's description and ingoing data ports are presented to the user; forms are generated based on the outgoing data ports of the human task (cf. Figure 5.14(b)). We decided to implement an application for the human task manager in the form of a mobile app as it enables the user to react to urgent requests in a timely and location independent manner on a personal mobile device [SLSS16]. However, arbitrary apps and devices could be connected to the WAMP-based WebSocket server to provide user interfaces for reacting to human tasks. Once the user filled in the forms and confirms the completion of the human task, the response is sent back to the PROtEUS system and incorporated into the overall process with the execution of the process instance continuing.

5.6.3. Process Manager

Besides the process modelling environment (IDE) presented in Section 4.8.1 [SHS16, SKNS15], a user interface/application to monitor and control process models and instances is an important component of a WfMS. Various stationary and mobile multi-modal applications exist to interact with PROtEUS. When providing means for ubiquitous human interaction with workflows, we focus on applications to support the development and management of CPS workflows and the corresponding WfMS with the help of mobile and stationary end-user devices. The automatic generation

of complex, adaptive user interfaces is out of scope as well as providing a high degree of user experience for end-users according to their capabilities and individual constraints. Users have to be able to interact with processes (i. e., to design, manage, control and monitor them) and they have to be able to manually interact as part of the process instance executions (i. e., by solving human tasks).

Desktop Application

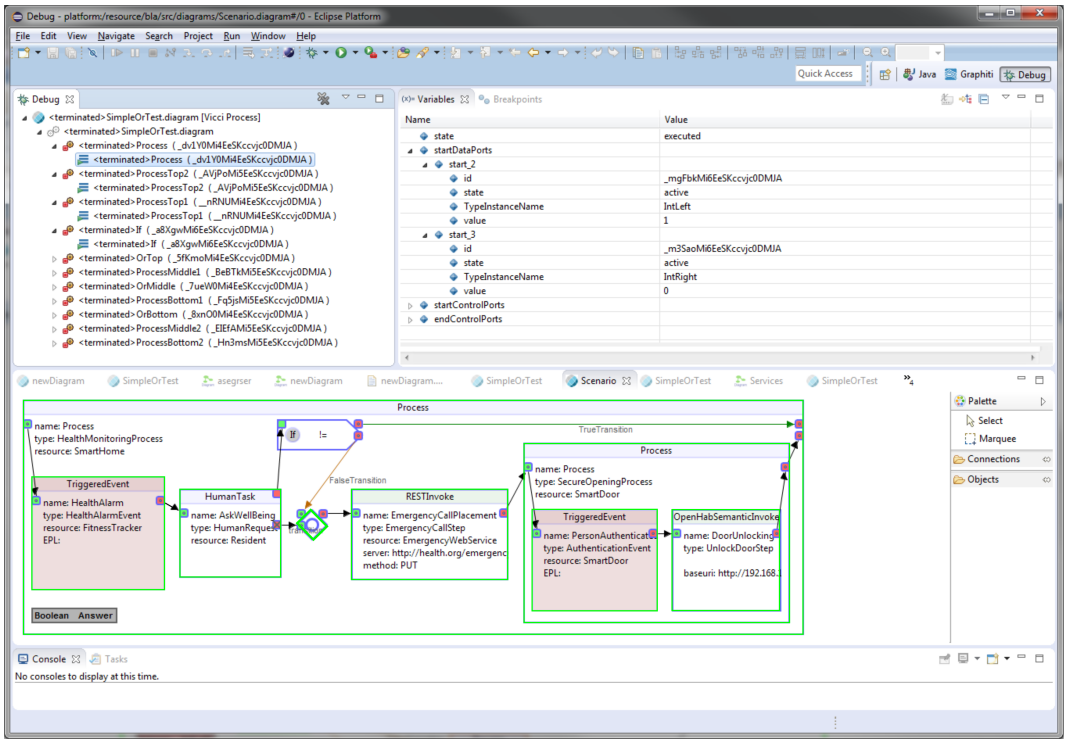


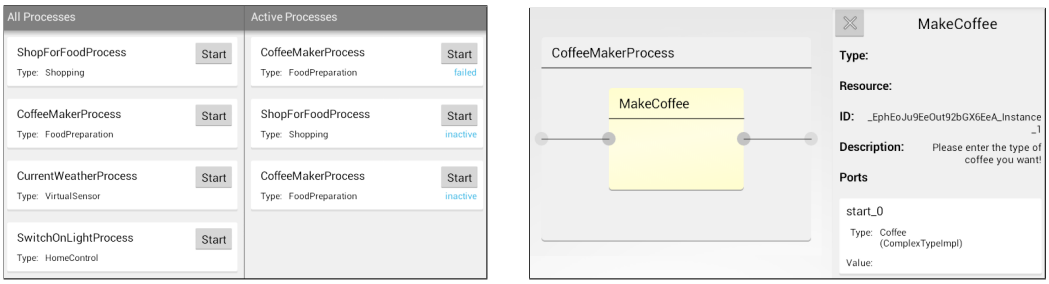
Figure 5.15.: Execution Trace for Emergency Scenario Process in the PROtEUS Desktop Application.

The PROtEUS desktop application is presented in Figure 5.15. Like the IDE, it uses Eclipse and EMF-based technologies to visualise and control the process execution (e. g., start/stop process instances) and to inspect current and historical process model and instance data. Eclipse's debugging view is used to present instance data regarding process step executions and data ports in the top of the window. The application is complemented by an instance viewer showing the graphical representation of selected active or finished process instances with highlighted execution traces (cf. bottom part of the Emergency scenario process instance in Figure 5.15).

Mobile Application

The mobile process management application is part of a larger Android-based CPS control app described in [SLSS16]. The *SmartCPS*[11] app provides a dashboard view

[11] `https://github.com/IoTUDresden/smartcps`



(a) Process Overview. (b) Process Details.

Figure 5.16.: Mobile Process Management App *SmartCPS*.

of important sensors, actuators and current process instances (cf. Figure 5.11). Besides a section dedicated to the management of human tasks, the app's process management section includes an overview and control of process instances and process models (cf. Figure 5.16(a)). Specific process models and instances can be inspected in detail in a dedicated view showing the graphical process model and model or instance information (cf. Figure 5.16(b)).

Tabletop Application

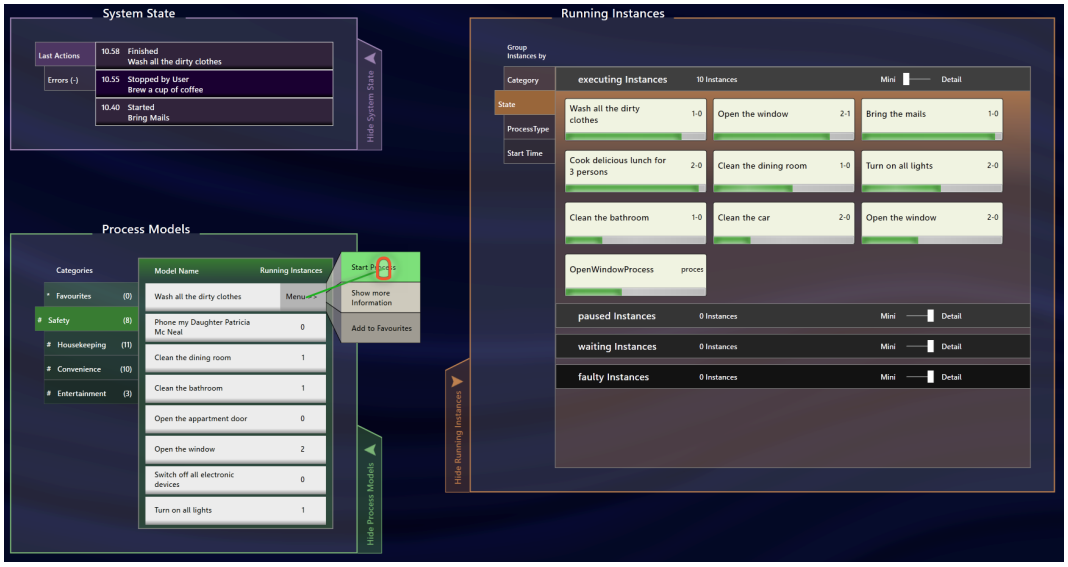
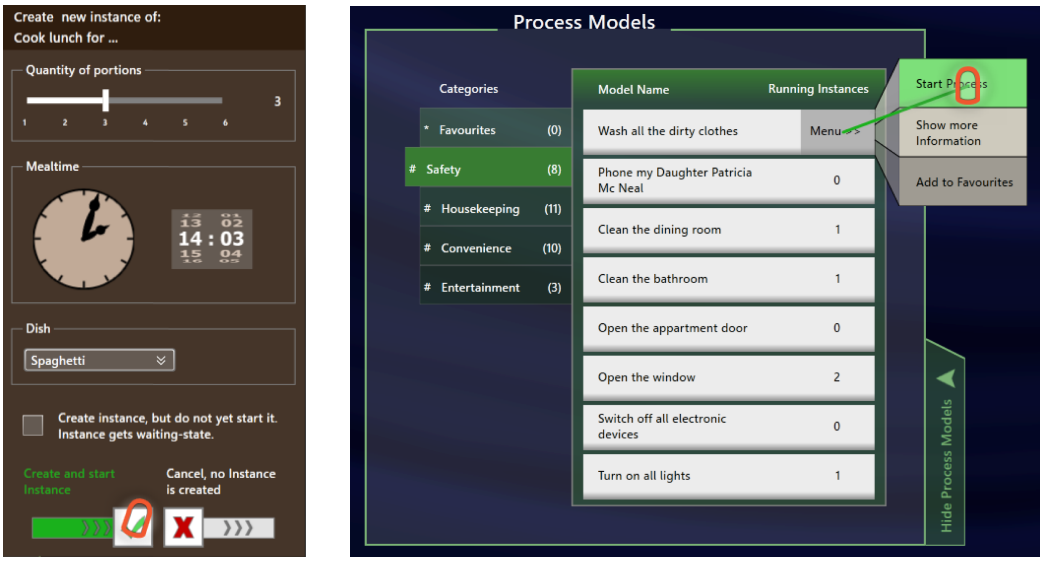


Figure 5.17.: Overview Screen of the Tabletop Process Management Application.

Following the idea of a having a control center application for managing all CPS devices and processes on a central stationary large screen computer, we developed a process management application for PROtEUS to be used on multi-touch tabletop devices as described in [SSMS14]. The *NatiaPro*[12] application presents a large overview screen of available process models, running instances and some general

[12] `https://github.com/IoTUDresden/NatiaPro`



(a) The Safe Select Gesture. (b) The LinMark Select Gesture.

Figure 5.18.: Special Gestures for Process Management on Tabletops.

system state information (cf. Figure 5.17). For every process model and instance, control options can be selected (cf. Figure 5.18(b)) and details opened in a new window (cf. Figure 5.18(a)).

The application is implemented on a tabletop, which uses optical finger/gesture and marker recognition (Samsung SUR40[13] with MicroSoft PixelSense technology) for multi-touch and tangible interaction. As this device's gesture recognition is very vulnerable and error prone with respect to light and other context factors, we developed a set of special gestures for controlling the application, which are more resilient against input errors and imprecisions. Management applications and their respective user interfaces for controlling CPS workflows have to be carefully designed with fault tolerance in mind as the manipulation of the workflow execution (start, stop, pause, etc.) also influences and affects processes and entities in the physical world. Input errors and undesired actions have to be avoided in this context.

One of the special process management gestures is the *LinMark Select* gesture for the safe selection of options from a list (cf. Figure 5.18(b)). In order to choose a concrete option of process-related actions, the user has to put down his/her finger and slide it to the desired option while holding down the finger. The *Safe Select* gesture is used for the confirmation of a process-related control action (cf. Figure 5.18(a)). The user has to swipe his/her finger along a specific slider to confirm the action to be executed. Both gestures are more complex than a simple tap, but also more resilient against errors as they require a longer interaction with the device and also consider the movement position and direction of the fingers to perform the interactions successfully. These gestures are still intuitive and easy to learn, though. With CPS also influencing the physical world, user interfaces have to be especially resilient against unintended control actions and other faulty behaviour. A short user study of the

[13] `https://www.samsung.com/ae/business/smart-signage/interactive-display-sur40/`

new gesture set showed a reduction of input errors when managing CPS workflows on tabletops [SSMS14].

Mixed Reality CPS Control

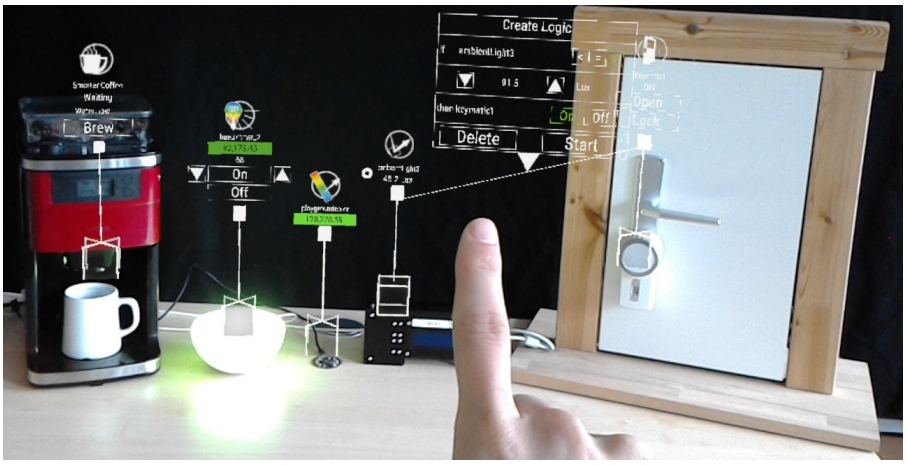
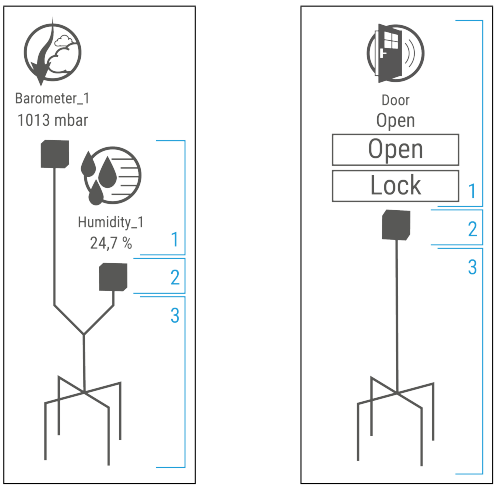


Figure 5.19.: Live View of HoloFlows.



(a) Sensors. (b) Actuators.

Figure 5.20.: Sensors and Actuators in *HoloFlows*.

The *HoloFlows* mixed reality application presented in Section 4.8.2 supports the direct manipulation of actuators according to their functionality, the displaying of current sensor values, as well as the visualisation and execution of workflows created with HoloFlows in augmented reality [SKGA17]. As shown in Figure 5.6.3, holograms are placed above the respective CPS device's physical location. Figure 5.19 shows the live view of a small smart home setup through the Microsoft HoloLens–including three actuators (door, lamp, coffee maker), two sensors (color, light), and a conditional sensor–actuator workflow between the light sensor and the door.

Sensor holograms include a type-specific icon, an identifier and their current value (cf. 1 in Figure 5.20(a)). They show a connector box for creating connections/workflows with other devices (cf. 2 in Figure 5.20(a)) and an anchor box for manual placement of the hologram in the holographic scene (cf. 3 in Figure 5.20(a)).

Actuator holograms also include a type-specific icon, an identifier, their current states and available control options that can be directly activated via an air tap gesture (cf. 1 in Figure 5.20(b)). They also show a connector box for creating connections/workflows with other devices (cf. 2 in Figure 5.20(b)) and an anchor box for manual placement of the hologram in the holographic scene (cf. 3 in Figure 5.20(b)).

5.7. Towards a CPS WfMS Reference Architecture for Other Domains

When designing the WfMS for CPS workflows, we followed the general suggestions of a reference architecture for WfMSes in [GdV98] and [DLRM+13]. PROtEUS features a core process engine for executing formalized process models, a process manager, process repository, external services, monitoring tools and user applications (cf. Sections 2.3.4 and 5.2). However, from the previous sections we conclude that with CPS, additional components become necessary to implement a fully functional workflow management system for CPS. The reference architecture has to be extended with components for interacting with new CPS devices–namely sensors and actuators–and with the physical environment. PROtEUS can be regarded as an implementation of this extended reference architecture.

Due to the possibly high number of sensors and other event sources in CPS and IoT, a complex event processing engine or a similar component for the processing of event streams is indispensable to allow for a fast event stream processing and context-sensitive behaviour related to multiple event sources and sensors. Service-based communication–either synchronous or asynchronous–is currently the de facto standard for communication with external IoT devices, which is why a component for invoking these services via remote procedure calls or publish/subscribe (event-driven) mechanisms has to be available as part of the CPS WfMS. However, this service invoker and the corresponding external services have to be relatively lightweight (e. g., relying on the REST paradigm) due to the limited availability of computing resources in CPS. Along with these limited resources and a high level of mobility there is a fluctuating availability of devices and services, which is why the underspecification and dynamic selection and assignment of services from a process has to be supported by the service invoker.

The process distribution component can be viewed as optional depending on whether the network structure supports this mechanism and this functionality is required for the respective domains and use cases. If there is a high number of CPS devices and CPS workflows and instances to be managed in large smart spaces, then a hierarchical overlay network of peers and super-peers executing process instances in a distributed way provides a scalable way to manage the process resources and executions. End-user applications have to be provided for designing and managing CPS workflows–not only on stationary desktop computers but also on mobile devices as these are the predominant class of interactive devices for future CPS and ubiquitous systems [Wei91]. This requirement also includes flexible and lightweight

(well-documented) bi-directional communication and programming interfaces to be provided by the CPS WfMS to interact with the WfMS and develop applications that use the WfMS programmatically.

Due to the large variety of possible CPS and IoT devices from various vendors providing heterogeneous programming and control interfaces, a middleware for unifying these CPS entities as well as for routing application calls to the respective devices has to be a dedicated component working independently of the WfMS due to its own complexity and use by other applications and systems. The middleware also has to provide service-based interfaces to interact with it and with its associated devices. Along with that middleware, a knowledge base containing the ontologically founded descriptions and relations of the CPS entities has to be available as dedicated external component to be accessed by the WfMS, middleware and other CPS applications for the purpose of dynamic resource selection.

The *Feedback Service* implementing a MAPE-K-based control and adaptation loop (cf. Section 6.2.1) also has to be considered as integral part of a more general reference architecture for a CPS WfMS to enable self-* capabilities. With new physical error sources and dynamic availability of resources, the workflow executions and WfMS have to be able to react autonomically to unanticipated situations as part of the self-management capabilities. The feedback loops should be executed implicitly as internal processes as corresponding process models having the MAPE-K steps modelled and executed explicitly would become very bloated and complex. With the requirement of being able to retrofit existing WfMSes with self-x capabilities, we propose to have a dedicated external component (*Feedback Service*) to implement the corresponding feedback loops as it can be reused with other information systems. More elaborations on this component and its applicability in other CPS domains can be found in Chapter 6.

In this thesis, we show the application and feasibility of the PROtEUS system and associated tools in **Smart Home** environments as examples for CPS. The applicability of the generalized system's structure as a reference architecture for other CPS domains remains to be further investigated. Many of the requirements of smart home environments can also be found in other domains (e. g., **Smart Offices** [FLSK13] or **Smart Factories** [SHH+14]) and vice versa, which is why the PROtEUS system as an implementation of a CPS WfMS reference architecture can be transferred to these domains and may prove suitable for executing workflows in other smart spaces (**Smart Buildings** [KA10]). With an active process distribution component and a multi-level peer–superpeer hierarchy, the system of workflow systems can also be scaled to the level of **Smart Cities** [PLM17]. All these environments have in common that they consist of a large number of sensors and resource constraint heterogeneous devices interacting with the physical world, which is why sensor event processing, dynamic resource selection, human interactions and resilient process execution are important requirements for these CPS domains, too. The generic concepts and approaches identified in related work (cf. Section 3.8) and applied within the PROtEUS system and the Feedback Service (cf. Chapter 6) have proven suitable to be used in various CPS domains and smart spaces as discussed in this thesis.

Some CPS domains introduce new requirements to the WfMS, which have to be evaluated further, though. The domain of **Industrial IoT** (Smart Production) imposes additional requirements on a corresponding CPS WfMS due to stricter real-

time and safety constraints [BS15]. The involved devices (production machines) often do not provide open programming interfaces or service-based access, which is why the WfMS has to be coupled tighter with the existing control applications (e. g., the SCADA system [DS99]) or using the emerging *OPC-UA* standard for industrial IoT [LM06]. The overheads regarding computations and communications introduced by a SOA often do not suffice the corresponding real-time constraints. Even stricter real-time and safety demands as well as closer software components and many parallel process instances influencing each other can be found in the **Avionics** and **Automotive** domains [LS16], which is why the suitability of our proposed concepts and reference architecture needs to be further evaluated. Here, real-time behaviour, safety guarantees/contracts as well as process execution verification have to be provided by the core process engine and its associated components. We do not discuss these aspects within the scope of this thesis. The proposed system architecture addresses the orchestration of workflow tasks at a more abstract higher level among multiple devices and across the boundaries of individual systems on the business process level. The execution of safety-critical and real-time demanding control actions is mostly left to the lower layers closer to the hardware within the particular system's or device's control software (cf. Section 2.5.1). Security and privacy are additional cross-cutting concerns not addressed in this thesis. However, they are important for many CPS domains (e. g., **Smart Hospitals** [HRZ15] or **Smart Mobility** [WBJ08]), which is why future developments and applications of our proposed CPS WfMS reference architecture also have to cover these issues.

6. Scalable Execution of Self-managed CPS Workflows

> "Any sufficiently advanced technology is indistinguishable from magic."
>
> Arthur C. Clarke

6.1. Introduction

Entities within CPS are characterized by being highly dynamic, resource constraint and also influenced by the physical world (cf. Section 2.4). This leads to new sources of errors, which the WfMS has to be able to handle in order to provide a resilient process execution in CPS (Requirements *R5–R7* in Section 2.6). In this chapter, we present an extension to the basic *PROtEUS* process execution system–the *Feedback Service*–that uses software-based feedback loops founded on the MAPE-K framework to verify the execution of activities within CPS workflows and to adapt the process resources in case of errors. Besides maintaining *Cyber-physical Consistency* (cf. Section 4.6), the Feedback Service is also capable of guaranteeing other QoS and KPI constraints based on the goal definitions for the individual process steps. Consistency levels and style sheets as well as the support of distributed processes lead to various dimensions of scalability that the CPS WfMS reaches. We show the exemplary application of the MAPE-K feedback loop in different workflows regarding cyber-physical process executions and the distributed execution of processes. Along with that, we discuss aspects related to the realization of cyber-physcial ACID properties and meta-adaptation for workflows. The Feedback Service can be used in combination with other existing WfMSes to add self-management capabilities to these systems (Requirement *R8*). We demonstrate the interaction of three existing WfMSes with the Feedback Service in a CPS workflow. The following elaborations are based on the information contained in [SHS16, SHHA16, SHS17, SHA17, Sei15, SHHA17, SHA18a].

6.2. MAPE-K Control Loops for Autonomous Workflows

The MAPE-K approach from autonomous computing serves as the basic concept to enable self-management for workflows in the context of this thesis. This feedback loop mechanism is the predominant technique to enable self-adaptation for CPS [MSW16]. Figure 6.1 shows our adaptation of the MAPE-K concept as described in Section 2.4.5 on the level of process steps. The *Managed Element* is a process step (here: the *SwitchOnLight* activity from our smart home example in

Section 2.2.1). Cyber-physical consistency (CPC) and other KPI and QoS criteria can be checked and ensured by using MAPE-K based feedback loops.

The basic idea of applying this concept on the process step level is to use sensor data from additional information sources to compare the effects of the execution of the process step instance with the expected outcome defined within the goals and objectives that are associated with the process step (cf. Section 4.5). With the increasing development of micro-electronics and its pervasion of almost every domain and object, new types of sensors, actuators and smart devices equipped with electronics to collect and communicate data emerge every day (*Internet of Things*). Many of these new sensors and actuators can be used to extend and retrofit existing devices, machines and objects with additional capabilities regarding data collection and other functionality. In comparison to the application on the control software layer, the consideration of these new data sources on the workflow layer (cf. Section 2.5.1) allows for an easier integration, reuse, extension and flexible adjustment of these new data sources within one or more processes (cf. Section 2.5.2). The following sections explain the individual phases and components of the MAPE-K-based autonomic manager for workflows in more detail.

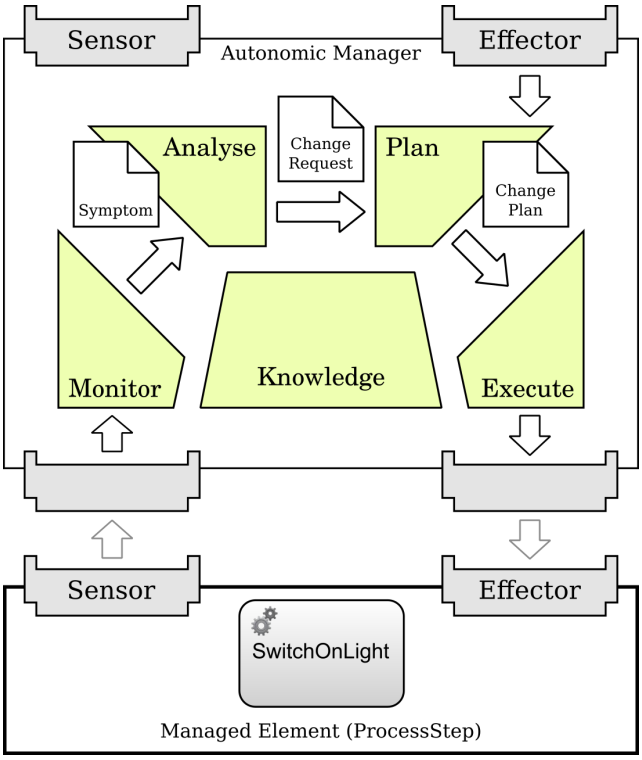


Figure 6.1.: Managing Process Steps with MAPE-K Feedback Loops in Accordance with [IBM05].

6.2.1. PROtEUS Extension: Feedback Service

The autonomic manager is implemented as a dedicated software component with a web service interface. This allows for a loose coupling of the *PROtEUS* process execution system and also other WfMSes with the autonomic manager component.

With the implementation of this component–called *Feedback Service*[1]–we follow a micro-service based approach to have a lightweight dedicated software component to provide the MAPE-K functionality [NSS14]. Figure 6.2 shows the extension of the PROtEUS base system and interaction with the Feedback Service. During enactment of a process instance, the goals defined for the respective process step are sent to the Feedback Service, which initializes the MAPE-K loop. While executing the loop(s)–described in the following sections–the Feedback Service interacts with sensors and actuators either directly via their service interfaces or via the middleware (cf. Sections 2.5.1 and 5.3). Once finished, the result of the feedback loop executions is published back to the PROtEUS system. The Feedback Service is composed of individual components for each phase of the MAPE-K loop that interact with each other. Figure 6.3 presents a detailed overview of the internal component structure of the Feedback Service and its internal and external interfaces.

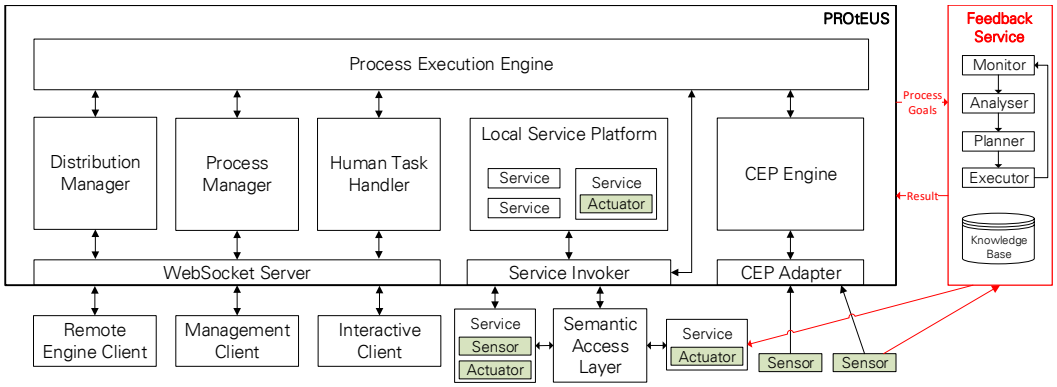


Figure 6.2.: The Basic Process Execution System (PROtEUS) and its Feedback Service Extensions.

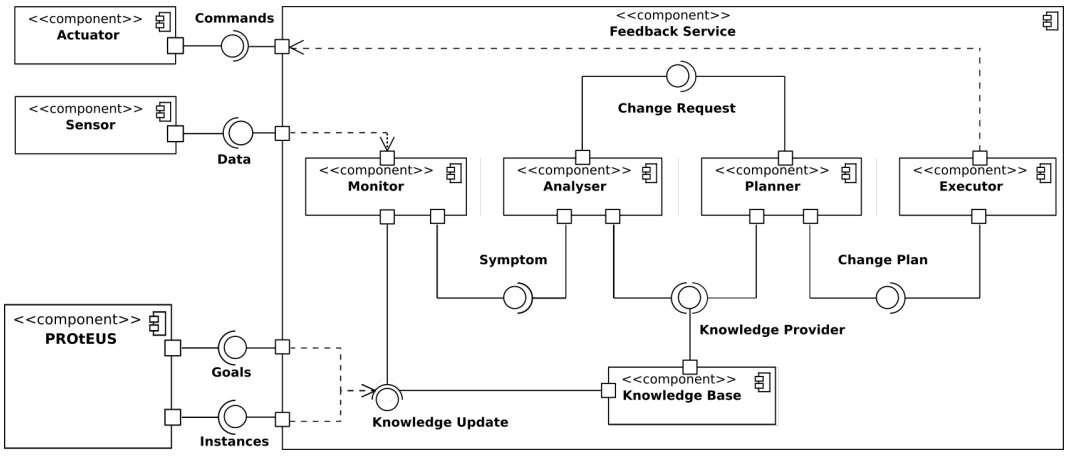


Figure 6.3.: The MAPE-K Feedback Loop as a Component-based Web Service Implementation.

[1] https://github.com/IoTUDresden/feedback-service

6.2.2. Knowledge Base

The *Knowledge (K) Base* is the central component to store and update all relevant process data as well as data concerning the CPS entities (sensors, actuators, objects, humans, software components, etc.) and their contexts. The structure and models used within the knowledge base are described in Section 4.3. The contents of the knowledge base are updated continuously at runtime during the execution of the MAPE-K phases (*Monitor*, *Plan*, *Analyse* and *Execute*), as well as during the enactment of process instances, and active connections to the IoT middleware.

6.2.3. Deployment and Instantiation

Depending on the availability of a Consistency Style Sheet for the respective process model (cf. Section 4.7), the Process Manager parses the style sheet during the deployment phase of a process (cf. Section 5.2.2) and sets the *cyber-physical* or *managed* properties as well as the goals and their objectives of the respective process steps. Otherwise, these attributes have to be contained in the process models. Upon instantiation of a process model by the PROtEUS process manager, the Process Engine executes instances of the modelled process steps and evaluates the *cyber-physical* or *managed* flags and *Goal* properties–either already set by the process modeller using the IDE or retrieved from the Consistency Style Sheet during deployment. In case these properties are set, the MAPE-K loop is initiated as part of the execution of the process step instance. A request containing the current goals of and information about the process step instance is sent to the Feedback Service in parallel to the execution of the particular process, process step, subprocess or activity by the PROtEUS system. The MAPE-K loop for this particular process step instance is then activated. The Feedback Service creates an internal workflow for every goal it receives that links the execution of the MAPE-K loops to the specific process step instances and the context data to be used from the knowledge base.

6.2.4. Monitor

In the *Monitor (M)* phase the dedicated *Monitor* component constantly updates the knowledge base with *Data* from *Sensors* and other components that it is subscribed to. A *Monitoring Agent* for each event source is used as an adaptor between the Feedback Service and the individual source. This agent subscribes to the events and unifies the received data to be stored in the knowledge base. During the monitoring phase, a pre-analysis of the data is performed to evaluate the significance of the data based on defined thresholds to filter jitter. Significant changes (*Symptoms* in Figure 6.1) within relevant context data are then fed forward into the *Analyser* component. The relevant context data is determined based on the *context path* definitions $CP = \{cp_1, ..., cp_n\}$ contained in the objectives $O = \{o_1, ..., o_n\}$ that belong to the goal Go of the respective process step (cf. Section 4.5). Only symptoms referring to these context paths are analysed in the next phase.

6.2.5. Analyse

In the *Analyse (A)* phase the *Analyser* component evaluates the received symptoms with respect to the *satisfied condition sc* and the *compensation condition cc*

as defined in the specific objectives–also considering the objective's *Consistency Level* L_O–of the process step. The satisfied condition combined with the optional consistency level defines a criterion for the successful execution of the respective process step. If this defined condition can be evaluated positively for every objective the process step contains, the step is assumed to be executed correctly. The Feedback Service reports this result back to the WfMS and the "regular" process instance execution continues. On the other hand, the criterion defined in the compensation condition indicates that an error may have occurred during the execution of the process step. In case this condition is evaluated positively for an objective, a change request regarding this particular objective is issued and the *Plan* phase is entered. If neither condition can be evaluated positively based on the symptoms received from the Monitor component, no further action is taken within this particular iteration of the MAPE-K loop. On receiving a new symptom from the Monitor, the Analyse phase is entered again.

The approach of evaluating sensor data from external sources to verify the process execution requires a stable and reliable sensor infrastructure. This topic is discussed in relation with business processes in [Wom11a, Wom11b]. We provide a remedy to possible errors in the sensor infrastructure by 1) having the possibility to define multiple objectives relating to different sensor nodes that can be analysed independently from each other; and 2) by using the *Cypher* query language [Web12] to define the respective context paths and the *SpEL* [JHA+13] to specify the compensation and satisfied conditions. Both languages allow for complex rule and query operations that can also pre-evaluate or filter data, e. g., calculate mean values of multiple sensors or limit the evaluations to certain time frames [CK11]. Although the PROtEUS base system already features a CEP engine to evaluate streams of complex sensor data, we decided to use a separate component to realize this functionality in the Feedback Service as it is intended to be a dedicated micro-service that can also be used in combination with other WfMSes. The redundant access to the sensors and actuators at this point separates the concerns of the basic CPS workflow execution by the WfMS and the self-management of the execution by the Feedback Service as an external supervisory component. A loose coupling of the CEP engine with the Feedback Service as well as the tighter integration of the CEP engine with the Feedback Service are subjects to future work. The application of more sophisticated means to synthesize goals and to determine derivations between the expected outcome of the execution and the actual outcome (e. g., by using machine learning, temporal logics or Situation Calculus as proposed in [MMS14]) can also be part of future research. Our approach is mostly based on rule definitions, which proofed feasible for our application scenarios (cf. Chapter 7). As the architecture of the Feedback Service is based on software components for each phase of the MAPE-K cycle, the implementation and with that, the concrete algorithms used within the Analyser can be easily exchanged by following the component interface definitions.

6.2.6. Plan

The *Plan (P)* phase executed by the *Planner* component is entered after a mismatch between the expected process step instance execution and the actual execution was determined based on a compensation condition defined in one of the objectives of the respective process step. The change request issued within the Analyse phase

and transferred to the Plan phase contains a simple description of the mismatch that occurred. Currently, for numerical context values, this mismatch corresponds to "too high" or "too low", for other non-numerical context factors to "unequal". The Planner component contains an extensible *Compensation Repository* and multiple *Compensation Queries* that are consulted in combination with the Knowledge Base to find a compensation for the occurred mismatch. Depending on the type of mismatch and use case, a specific compensation action (cf. Section 4.5) is derived by the Planner. We show concrete examples of determining these compensation actions for *cyber-physical* process steps in Section 6.3 and for *distributed* processes in Section 6.4. In general, the Planner looks for alternative process resources in the same context of the original resource or of the related sensor node defined in the respective context path [CL08]. Depending on the determined mismatch, the process step could be repeated by the alternative resource or the commands for increasing or decreasing the respective numerical values are selected from the Knowledge Base to be executed. This *Change Plan* is then transferred to the *Execute* phase. In case the Planner is not successful with deriving a compensation strategy (e. g., there are no available replacement resources), the *Error Case* is entered.

In order to derive a suitable compensation action in a more sophisticated way, a detailed classification (ontology) of possible errors and corresponding remedies is necessary. We currently use a simple rule/query-based approach to find alternative resources to execute the erroneous process steps–either to repeat them or to execute compensating actions for simple numerical mismatches. Especially in the context of CPS, a repetition of the same action in the physical world or a simple rollback of an action is not possible to due the complexity and limitations of the physical world. For that reason, a domain-specific description of error cases and remedies (remedy workflows) has to be developed and integrated into the compensation repository similar to *Exlets* proposed in [ATHVDAE07]. More advanced approaches for deriving compensation actions exist (e. g,. using Case-based Reasoning [WRWR05], Classical Planning [MMS14] or other inference-based mechanism involving the SAL [HSKS16b]), but are out of scope of this work. As the adaptation strategies have to be derived at runtime, inference-based mechanisms may lack the necessary performance to be applied in CPS [Lee08]. Similar to the Analyser component, the actual implementations of the Planner component can be exchanged. A tighter integration of the mechanisms for dynamic service discovery provided by the SAL with the Feedback Service and a shared knowledge base for both components in the Plan phase remains subject to future work.

6.2.7. Execute

In the *Execute (E)* phase the *Executor* component enacts the change plan received from the Planner. This change plan contains the *Commands/Compensation Actions* to be executed in order to compensate the occurred mismatch. In our current implementation, this includes the selection of the appropriate resource (*Actuator*) and executing the derived operation (e. g., *ON*, *OFF*, *UP*, *DOWN*), which is a REST call to the service that represents the compensation operation as defined in the Knowledge Base. Sections 6.3 and 6.4 show this process in more detail. During the execution of the compensation action, the MAPE-K loop is used again to check if the corresponding objective has been fulfilled with respect to the satisfied condition

of the particular process step. At this point, the consistency level defined complementary to the satisfied condition may reduce the number of necessary MAPE-K iterations necessary to restore consistency. If the objective can be evaluated positively in subsequent feedback loops, the Feedback Service terminates the MAPE-K cycle for the objective in question. In case all objectives belonging to a process step can be fulfilled this way, a positive result is reported back to the WfMS and the process execution continues. Due to the composite structure of our process metamodel, the MAPE-K mechanism can be applied on various process levels: on atomic process steps (activities), subprocesses or entire processes. This probably results in multiple hierarchical MAPE-K loops running in parallel. It is up to the workflow designer to specify reasonable goals and objectives for the process steps that will not influence each other negatively.

With our approach, we focus on the adaptation of the process resources as the main new error sources in CPS. However, rule-based and structural adaptations of workflows [RBA08, DR09, RRKD05, MGR04], the synthesis of new process steps [MMS14, RSA10] or the replacement of activities or subprocesses [Sch08] can also be adaptation strategies to be considered for future work. We currently follow the approach of adapting the process instances in case of errors, creating a form of ad-hoc change to increase the resilience of a process activity (*ad-hoc workflows*). With reoccurring deviations of the process executions from the expected outcome for multiple instances of the same process step, the adaptation of the corresponding process model could also be an option to improve the process quality (*Process Evolution* [RWRW05]).

6.2.8. Error Case

During the *Plan* phase, the Planner component tries to derive compensation actions suitable to remedy the occurred error. This process relies on information contained in the Knowledge Base regarding available instances of process resources and their functionality. In case the enactment of the compensation actions does not lead to the fulfilment of the corresponding objective or no alternative process resource can be found, the execution of the MAPE-K loop terminates for the objective and an error is reported back to the WfMS (*Error Case*). In our implementation, this will activate a special *Failure Port* (cf. Section 4.2.7) and the corresponding error branch that was modelled by the process designer. The process execution continues along the error branch and the main process branch is deactivated. It is up to the process designer to define the process steps contained in this branch. It could include a *Human Task* (cf. Section 4.2.10) asking the user to take care of the problem or recommend the execution of some manual tasks to solve the issue. After the user confirms that the error has been fixed, which could again be monitored through a feedback loop for the Human Task, the process could be designed to have an exclusive gateway to merge with the main branch of the process again. The WfMS's internal exception handling mechanisms could be activated too to try to resolve the errors the Feedback Service is not able to compensate or other advanced approaches for exception handling in workflows (e. g., *Exlets* [ATHVDAE07]) could be applied.

Listing 6.1: Goal and Objective for SwitchOnLight Process Step.

```
1 "SwitchOnLight" : {
2  "name": "enough light for working",
3  "objectives": [
4   { "name": "kitchen light > 700 lux in 5 seconds",
5     "satisfiedCondition": "#lightIntensity > 700",
6     "compensationCondition": "#objective.created.isBefore
         (#now.minusSeconds(5))",
7     "contextPaths": [
8      "MATCH (thing)-[:type]->(sensor {name: 'LightSensor
          '})",
9          "MATCH (thing)-[:isIn]->(room {name: '
             Kitchen_Mueller'})",
10         "MATCH (thing)-[:hasState]->(state:
             LightIntensityState)",
11         "MATCH (state)-[:hasStateValue]->(value)",
12         "WHERE toFloat(value.realStateValue) > 0",
13         "RETURN avg(toFloat(value.realStateValue)) AS
             lightIntensity"
14 ] } ] }
```

6.3. Feedback Loop for Cyber-physical Consistency

In this section we show the application of the MAPE-K-based feedback loop using the *Feedback Service* to check and restore *Cyber-physical Consistency* (CPC) [SHHA16] for the smart lighting process step of the *Morning Routine* scenario process (cf. Section 2.2.1). The managed elements are the *SwitchOnLight* process step and the corresponding process resource (cf. Figure 6.1). Additional scenarios and applications of the MAPE-K loop to maintain CPC can be found in Chapter 7.

6.3.1. Knowledge Base

This scenario refers to the specific sensor and actuator models contained in the knowledge base. The involved sensors are light sensors to measure the light levels (cf. Figure 4.11) and the actuators are dimmer switches to control the lights (cf. Figure 4.12).

6.3.2. Goal and Objective

The corresponding goal definition written as a Cypher query can be found in Listing 6.1. The goal "enough light for working" (Line 2) contains the objective "kitchen light $>$ 700 lux in 5 seconds" (Line 4). The corresponding satisfied condition (Line 5) defines light levels in the kitchen above 700 Lux as success criterion. If these levels are not reached within 5 seconds after the execution of the basic process activity, then an error is assumed and a compensation is searched for (Line 6). The context path retrieves the light levels of sensors of type "LightSensor" in room "Kitchen_Mueller" (Lines 7–13). The execution of the MAPE-K feedback loops for this scenario is depicted in Figure 6.4.

6.3.3. Deployment and Instantiation

The Process Engine executes the process instance containing the process step of type *RESTInvoke* in the regular way. It initiates the *Service Caller* as a client to invoke the specific REST method on the resource (web server) or middleware controlling *DimmerSwitch1*. The web server reports the successful execution back to the Process Engine. For this process step, the *cyber-physical* flag is set and its *Goal* attribute corresponds to the contents of Listing 6.1. Consequently, the *Feedback Service* (FB Service) is called from the Process Engine in parallel to the REST service call to the dimmer switch. The FB Service receives the goal and instantiates the MAPE-K loop to be run in parallel to verify the execution of the process step instance for the specific objective.

6.3.4. Monitor

The *Knowledge Base* KB is subscribed to all available sensors. The *Light Sensor* in question updates its values about once every second. Upon initiation of the MAPE-K loop, the *Monitor* subscribes to the KB with respect to the light sensor values c_t as defined in the context path *cp* (cf. Section 4.8.3). From that point on, it receives all updates concerning the sensor from the KB. With every significant update (Symptom) of the light levels from the sensor, the corresponding monitoring agent sends a request to the *Analyser*.

6.3.5. Analyse

The *Analyser* checks the compensation condition and satisfied condition w. r. t. the received symptoms. As in our example the light levels from the sensor do not exceed the defined threshold of 700 Lux, the Analyser triggers a timeout after 5 seconds as defined in the compensation condition. At this point in time, we assume that CPC_t is violated due to the assumed state $S_{C,t}$ regarding the current light levels $c_t^{'}$ of DimmerSwitch1 (*On*) not matching the actual state $S_{P,t}$ (*Off*) and light levels c_t (cf. Equation 4.2). A *Change Request* is then issued and transferred to the *Planner*.

6.3.6. Plan

The Analyser determined that the mismatch regarding the relevant light levels is *too low*. Based on this mismatch and the context of the MAPE-K execution (checking for *Cyber-physical Consistency CPC*), the *Planner* queries the *Compensation Repository* for a suitable compensation strategy. Parts of this repository are *Compensation Queries* to be executed for the knowledge base. The compensation query concerning *CPC* restoration can be found in Listing 6.2.

The essence of this query is that the Planner looks for actuators in the same room (here: Kitchen_Mueller) as the light sensor is in (Line 5). The devices–actuators and sensors–have to be linked to the same context attributes (here: light levels) (Line 6). The rest of the query is used to retrieve the actuator (here: DimmerSwitch2) based on its functionality and with that, the control commands to influence the context attribute (Lines 7–20) as modelled and contained in the Knowledge Base. Based on the *too low* mismatch, the Planner then selects the found compensation action (here: *UP*) to increase the value of the context attribute (here: light levels). The

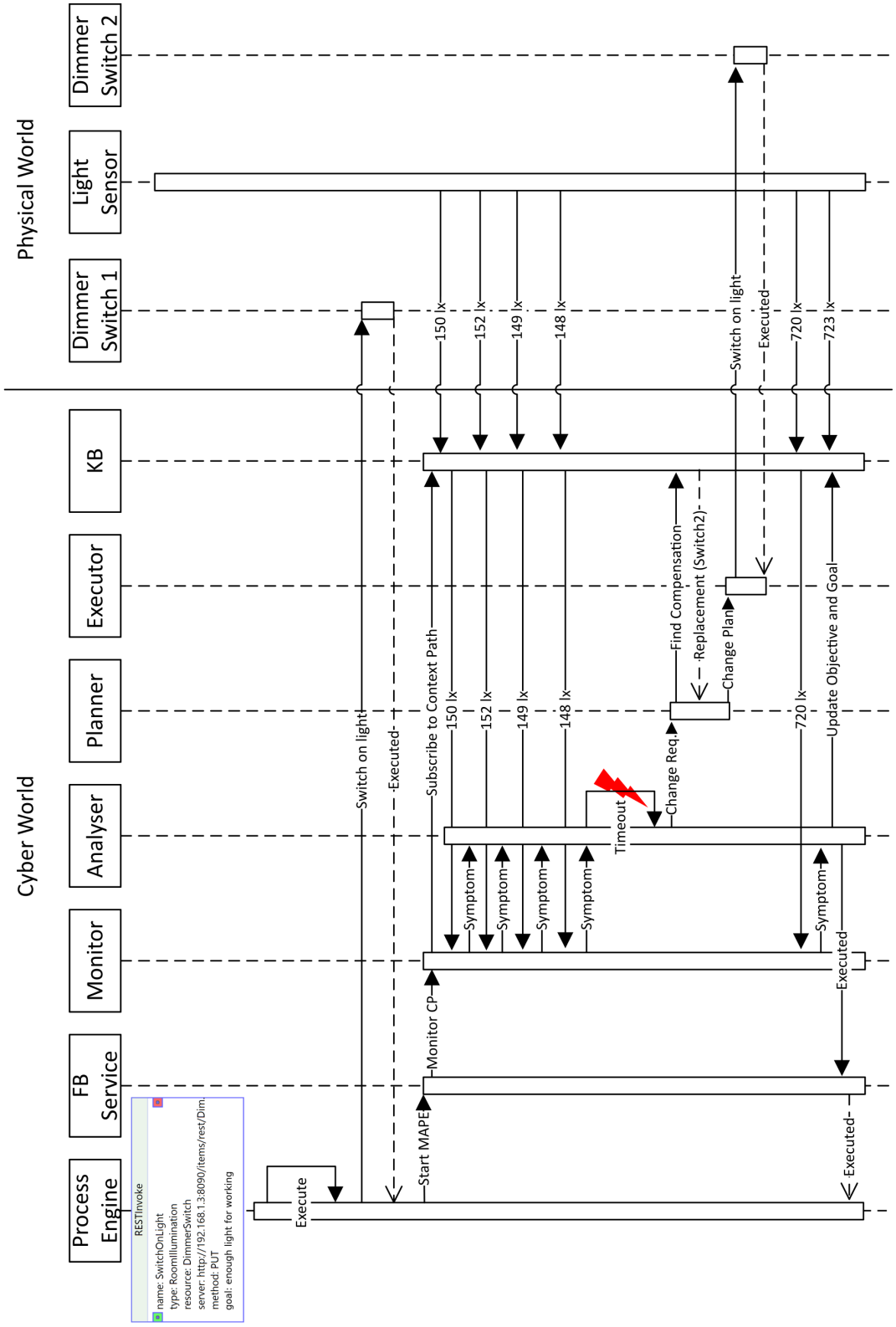


Figure 6.4.: Sequence Chart for the MAPE-K Loop applied to the Smart Lighting Process Step.

Listing 6.2: Compensation Query regarding the Restoration of Cyber-physical Consistency.

```
1  MATCH (sensor)-[:hasState]->(sensorState)
2  MATCH (actuator)-[:hasState]->(actuatorState)
3  MATCH (actuatorState)-[:hasStateValue]->(
       actuatorStateValue)
4  MATCH (actuatorStateValue)-[:unitOfMeasure]->(
       actuatorStateUnit)
5  MATCH (sensor)-[:isIn]->(room)<-[:isIn]-(actuator)
6  MATCH (sensorState)-[:type]->()-[:subClassOf*0..1]->()
       <-[:type]-(actuatorState)
7  MATCH (actuator)-[:hasFunctionality]->(function)-[:
       hasCommand]->(command)
8  MATCH (command)-[:type]->(commandType)
9  MATCH (function)-[:type]->()-[:subClassOf*]->({ name: "
       ControlFunctionality" })
10
11 WHERE id(sensorState) = {stateId} AND has(command.
       realCommandName) AND has(commandType.name)
12
13 RETURN DISTINCT
14         actuatorState.name AS actuator,
15         commandType.name AS commandType,
16         command.realCommandName AS commandName,
17         actuatorStateValue.realStateValue AS
              actuatorState,
18         actuatorStateUnit.name AS actuatorStateUnit
```

reference to the command including the respective service URI to be invoked is then transferred as part of the *Change Plan* to the *Executor*.

6.3.7. Execute

The *Executor* invokes the corresponding command contained in the Change Plan to activate DimmerSwitch2. The Monitor is still subscribed to the light levels as defined in the context path. It receives the changed light levels, which are analysed again with respect to the fulfilment of the compensation condition and satisfied condition. The current light levels now match the criterion defined in the satisfied condition leading to the termination of the MAPE-K loop and Feedback Service. Before terminating, the Analyser updates the data related to the MAPE-K executions (goals and objectives) in the knowledge base. Figure 4.17 shows the corresponding entries. Cyber-physical Consistency is restored with the positive evaluation of the satisfied condition and therefore, the fulfilment of objective and goal associated with the process step is achieved.

6.4. Feedback Loop for Distributed Workflows

In order to illustrate the applicability of the proposed MAPE-K framework and Feedback Service component in other contexts, we apply the framework to the distributed execution of subprocesses on a mobile service robot (Turtlebot) as execution peer [SHA17]. Figure 6.5 shows the adaptation of the Autonomic Manager to manage the process resource (*Turtlebot*) executing a subprocess. The exemplary subprocess "Retrieve Insulin Injection" stems from the process shown in Figure 5.10. This subprocess consists of multiple process steps instructing the service robot (Turtlebot) to drive to different destinations. The successful execution of this subprocess is rather crucial, but the robot is vulnerable to various physical and technical errors and obstacles, e. g., the drainage of its own battery, the navigation algorithms cancelling due to unknown obstacles or loss of orientation, or the loss of WiFi connectivity. In this example, we show the execution of the MAPE-K feedback loop to check the reachability of the mobile robot and execute compensating actions in case it gets disconnected from the network. This application of the MAPE-K mechanism shows that, besides criteria concerning Cyber-physical Consistency, also QoS and arbitrary other constraints can be defined to verify and possibly adapt the process execution with our suggestion of using the MAPE-K feedback loop for self-management of workflows. Additional scenarios and applications of the MAPE-K principles in the context of distributed process executions can be found in Chapter 7.

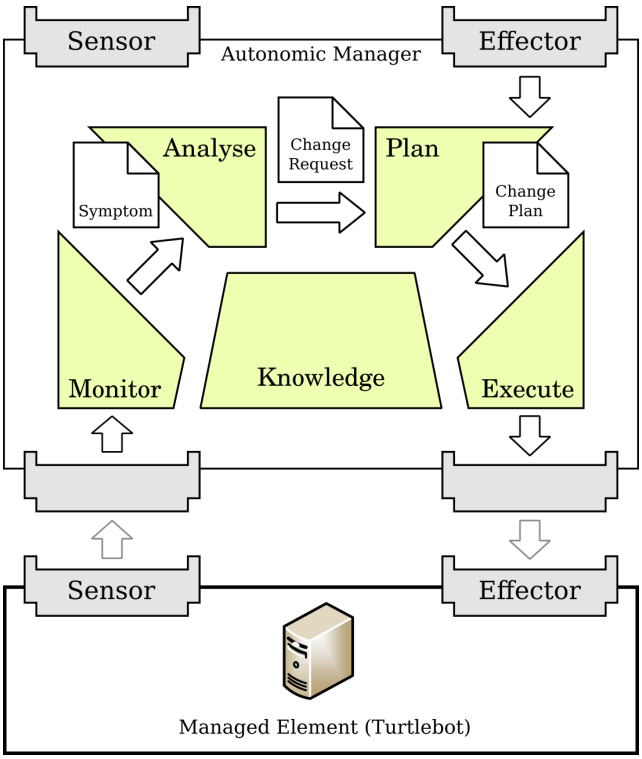


Figure 6.5.: Managing Distributed Process Resources with MAPE-K Loops.

6.4.1. Knowledge Base

In this scenario, we use the context model from the Knowledge Base (KB) that refers to the Turtlebot robot as presented in Section 4.3.3. The robot acts as a peer in the peer–super-peer architecture (cf. Section 5.5.1). It is able to execute instances of process steps due to PROtEUS running on the robot and it publishes various runtime metrics (e. g., last heartbeat and battery levels). The KB contains information regarding super-peers and its associated peers as well as the process instances that are running on super-peers and peers.

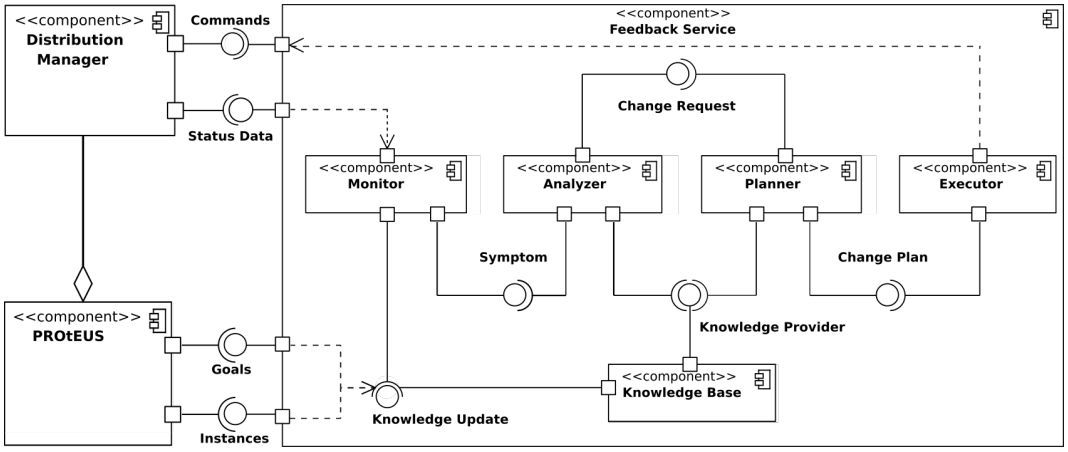


Figure 6.6.: Interaction between PROtEUS and the Feedback Service to Execute Distributed Processes.

6.4.2. Goal and Objective

Listing 6.3 shows the goal and objective for this use case. It refers to the point of time of the robots last liveliness signal (heartbeat) and the execution state of the particular subprocess. The satisfied condition states that the execution went well if the subprocess's state is "executed" (Line 5). On the other hand, the compensation condition states that there is a need for initiating the search for a compensation when the last heartbeat was longer than 5 seconds ago and the subprocess was still "executing", i. e., the robot is very likely to have lost its network connection while executing the process instance (Line 6). The context path is specified to retrieve the peer's (Turtlebot1) last heartbeat and the execution state of the subprocess (Lines 7–9). The identifier of the subprocess instance is used to link the instance to the specific execution peer (*RUNS_ON* relation). Figure 6.6 shows the interaction between the PROtEUS components and the Feedback Service for this setup. The execution of the MAPE-K feedback loops for this scenario is depicted in Figure 6.7.

6.4.3. Deployment and Instantiation

The overall process is modelled in accordance with Figure 5.10. "Turtlebot1" is specified as the responsible process resource and the goal modelled for the "RetrieveInsulinInjection" subprocess corresponds to Listing 6.3. As this example process is

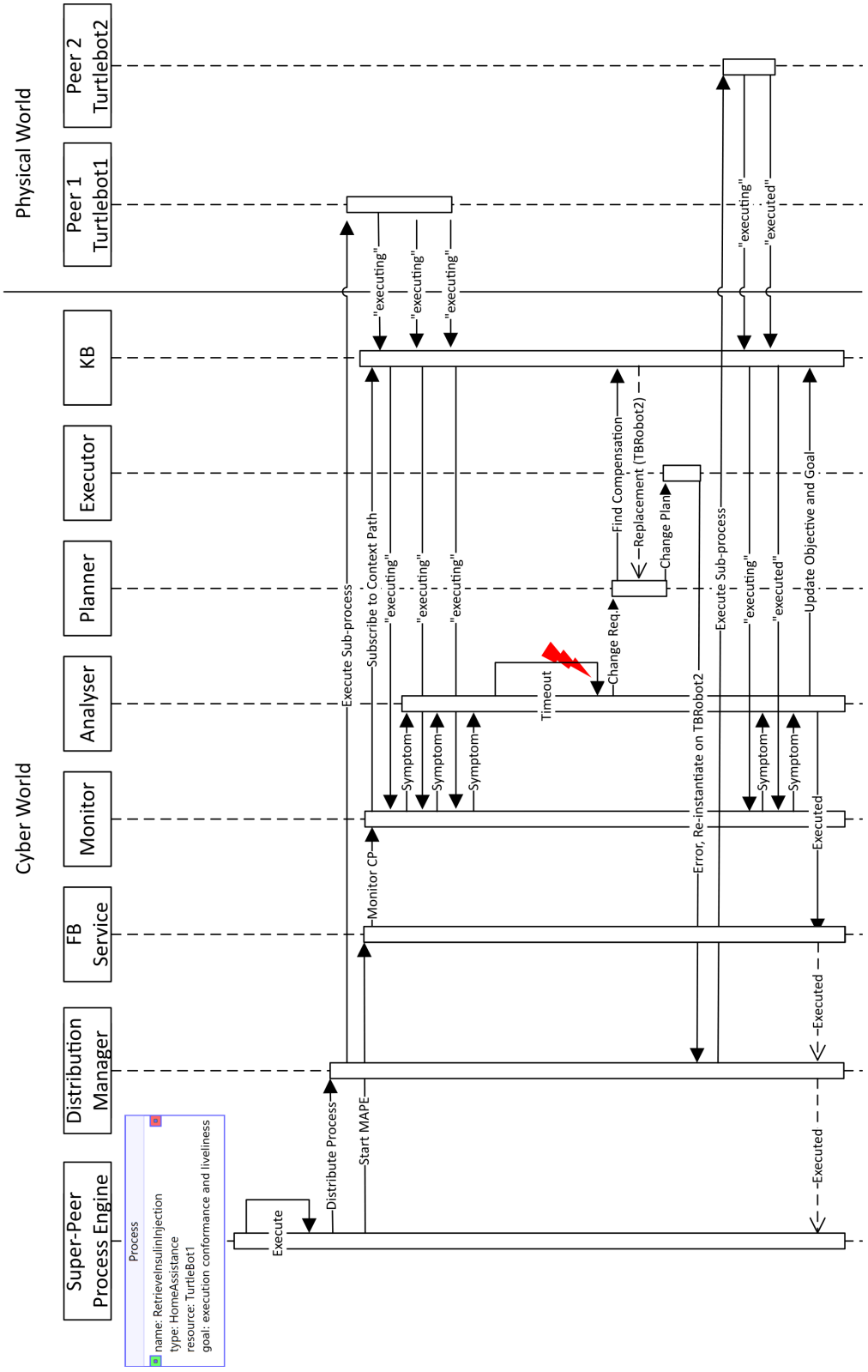


Figure 6.7.: Sequence Chart for the MAPE-K Loop applied to the Distributed Robot Process.

Listing 6.3: Goal and Objective for Distributed Subprocess Execution on Turtlebot.

```
"RetrieveInsulinInjection": {
 "name":"execution conformance and liveliness",
 "objectives" : [
 { "name":"heartbeat < 5 seconds and executed",
   "satisfiedCondition": "#processState == 'executed'",
   "compensationCondition": "#timeFrom(#heartBeat).
      isBefore(#now.minusSeconds(5)) and #processState
      == 'executing'"
   "contextPaths": [
    "MATCH(n:NeoProcess {processId:'
       RetrieveInsulinInjection'})-[r:RUNS_ON]->(p:
       NeoPeer)",
        "RETURN n.state AS processState, p.lastHeartbeat
           AS heartBeat"
] } ] }
```

executed as a distributed process, the PROtEUS workflow system on the super-peer (*D-PROtEUS*) instantiates the main process and begins its execution. Once it reaches the subprocess in question, it evaluates the process resource attribute, searches for the corresponding peer and the *Distribution Manager* of the super-peer sends the subprocess to *Peer1* (Turtlebot1). This requires the Turtlebot1 peer to be connected to and registered with the super-peer and the PROtEUS WfMS also to be running on this peer. In parallel to sending the request to execute an instance of the subprocess to the peer, the super-peer D-PROtEUS system invokes the Feedback Service (FB Service) with the goal to execute the MAPE-K loop for this distributed subprocess instance, which is marked as *managed* process step (cf. Section 4.5.2).

6.4.4. Monitor

Upon receiving the request from the super-peer, the Feedback Service starts monitoring the data from the Knowledge Base (KB) as defined in the context path (CP). Peer1 sends periodic status updates regarding the state of the subprocess execution. This information is used to update the process execution related data ("processState") contained in the KB and the message's timestamp is used to set the "heartBeat" property of the peer to indicate its last liveliness signal. Both values are monitored by the FB Service. With every relevant update of these values (symptoms), the Analyser is triggered to evaluate the data.

6.4.5. Analyse

The Analyser processes the symptoms with respect to the compensation condition and satisfied condition. It will not initiate any actions if the process is in state "executing" and the last liveliness signal was received less than 5 seconds ago. Figure 6.7 shows that after sending three status updates, the Turtlebot1 peer is not publishing messages anymore. Eventually, this leads to the Analyser sending a change request to search for a compensation to the Planner as the compensation condition becomes

Listing 6.4: Compensation Query regarding Distributed Process Execution.

```
1  Match(original:NeoProcess)
2  WHERE ID(original) = {processNodeId}
3  WITH original
4
5  Match(remote:NeoProcess)-[remoteFor:REMOTE_FOR]->(
       original)
6  WITH remote, original
7
8  Match(original)-[runsOnSuper:RUNS_ON]->(originalPeer:
       NeoPeer)
9  WITH originalPeer, remote, original
10
11 Match(remote)-[runsOnRemote:RUNS_ON]->(remotePeer:NeoPeer
       )
12 WITH remotePeer, originalPeer, remote, original
13
14 MATCH (newPeer:NeoPeer)
15 WHERE newPeer <> originalPeer AND newPeer <> remotePeer
16
17 RETURN original, remotePeer, newPeer
```

true after 5 seconds of not receiving any new symptoms and the process still being in state "executing".

6.4.6. Plan

In the context of a peer failing during the distributed execution of a subprocess instance, the Planner searches for an alternative peer registered with the super-peer to repeat the execution of the subprocess instance. The compensation repository contains the compensation query shown in Listing 6.4 that will be executed in this context. From the data related to the subprocess execution on the peer (Lines 5–6), it basically tries to determine the corresponding super-peer and main process on the super-peer (Lines 8–9). It then tries to find a new peer for execution that is distinct from the super-peer and the failed peer (Lines 11–17). This alternative peer (*Turtlebot2*) is transferred as part of the Change Plan to the Executor.

The strategy of selecting an alternative peer to repeat the execution in case the original peer fails is used here for the purpose of simplifying the explanations. More sophisticated strategies and compensation queries regarding the capabilities of the respective devices and also the progress of the process execution on the failed peer should be considered for this and other cyber-physical scenarios. If Turtlebot1 already retrieved the injection and fails on its way to the resident, then the repetition of the entire subprocess by Turtlebot2 could cause additional problems as there may not be a second injection available in the shelf. As already discussed in Section 6.2.6, a more sophisticated classification of errors and corresponding compensation strategies needs to be developed to ensure the successful execution of processes and feedback loops in CPS.

6.4.7. Execute

Based on the Change Plan received from the Planner, the Executor informs the Distribution Manager about the error that occurred during the execution of the subprocess on Turtlebot1. It also submits the URI of the alternative peer to reinstantiate the subprocess on. The Distribution Manager invokes the Process Manager of the PROtEUS instance running on Turtlebot2 to execute the subprocess again. The Monitor is still active from its initiation by the super-peer and listens to updates regarding the execution of the specific "RetrieveInsulinInjection" process. Once the Turtlebot2 peer starts executing the process, the Monitor receives updates again that are subsequently evaluated by the Analyser. The execution on Turtlebot2 is successful–the Analyser receives the "executed" state for the subprocess without reaching a timeout. Hence, the satisfied condition becomes true and the states of the objective and goal in the Knowledge Base are updated. This then leads to the termination of the feedback loop for this subprocess instance and the execution of the main process continues on the super-peer.

6.5. Consistency Levels, Scalability and Scalable Consistency

The number of repetitions of the MAPE-K loop can be adjusted by defining or modifying the *Consistency Level* for the respective satisfied conditions contained within the process goal's objectives. For mismatches regarding numeric context values–physical or virtual–we search for *Increase* or *Decrease* operations in the planning phase that are able to influence the values accordingly. For the lighting example, this could mean a stepwise increase or decrease of a dimmer switch's power levels (e. g., by 10 %) until the threshold defined by the consistency level is reached. By scaling this level up or down, the number of necessary MAPE-K iterations can be scaled resulting in *Scalable Consistency*. This leads to an increase or decrease of the execution time based on the number of necessary MAPE-K loops to fulfil the consistency/precision requirements as defined by the consistency level. In general, there is a direct proportionality between the consistency level and the number of MAPE-K iterations and with that, the overall execution time. The lower the required precision (consistency level), the lower the potential overall execution time (number of MAPE-K iterations)–and vice versa. By this, the execution of self-managed process steps becomes scalable with respect to the required level of precision, which is depending on the concrete use case, workflow and domain (cf. Section 4.6).

The separation of concerns with respect to the goal definitions in *Consistency Style Sheets* and the modelling of the *basic* workflow in the "traditional" way also contributes to the scalability of our approach. The definition of cyber-physical workflows and tweaking of their parameters and goals is often a cumbersome process that relies on many experiments and various repetitions and variations of workflows to find optimal configurations. The basic workflow defining the "base recipe" is specified using the regular underlying workflow notation. The fine-tuning of more unreliable, imprecise and vulnerable (physical) parameters is then done with the help of the Consistency Style Sheets. This facilitates the execution of real-world experiments

as only the parameters in the style sheet have to be modified and the workflow can be repeated again more efficiently and at a larger scale.

The distributed execution of processes in the peer–super-peer hierarchy also allows for scaling up the process execution with respect to the number of devices involved in the process execution and the number of parallel process instances. With an increased number of available peers that are possibly also specialized in executing certain kinds of subprocesses (e. g., computing intensive tasks), more resources are available in the CPS to execute the processes. Super-peers can take over the tasks of managing the process executions and all other IoT devices may act as individual peers running the *bare* WfMS to execute subprocesses. Implementing multi-level hierarchies of peer–super-peer networks allows for an efficient management and scaling of the number of CPS devices as resources and of the workflow execution up to the level of smart cities. An exemplary setup related to a smart city may view the devices of a single room or floor of a house as peers, and one super-peer per room or floor orchestrating/managing the processes within the room/floor and communicating with other super-peers managing the process execution in other rooms/floors. These super-peers then act as peers to the super-peer managing the processes within the smart building. This super-peer can then again play the role of a subordinate peer to a higher-level super-peer orchestrating processes in a smart neighbourhood. This can be continued to the scale of smart districts and smart cities. While this discussion seems plausible on a theoretical level, the corresponding experiments need be conducted within an adequate city-wide scale to evaluate the feasibility of the hierarchical peer–super-peer concepts as part of future work.

6.6. Self-managed Workflows

The examples discussed in this chapter show how to react to a certain degree of unanticipated situations and provide a basic *self-healing* mechanism for CPS workflows. Assuming the process designer knows the relevant expected outcome of the execution of a process activity, the system is able to check and verify if the execution was successful by analysing additional sensor information from arbitrary sources (*self-awareness*). Goals and objectives hereby provide a more declarative and flexible approach as we do not need to explicitly specify the reaction to every possible error scenario as part of the process model in advance. This modelling of possible errors and their remedies as explicit branches within a process would lead to very complex process specifications that may not be very flexible and hard to redesign or adapt. Based on measurable mismatches and the goal specifications, the Feedback Service can derive simple known compensation actions from the compensation repository to remedy the occurred errors or derivations.

Within our scenarios, we focus on the adaptation of process resources as the main sources of errors and imprecision when interacting with the physical world. However, besides the self-healing with respect to process resources, goal specifications may also comprise other factors and parameters, e. g., throughput and latency of certain process executions or the overall energy consumption [SGCG18] to address other self-* properties. If the Knowledge Base contains these values and increase/decrease operations linked to these QoS or KPI factors are available, then the Feedback Service can also be used for other purposes regarding the implementation of additional self-*

mechanisms, e. g., *self-optimization*, *self-configuration* or *self-management* of workflows in general (cf. Section 2.4.5). We investigate these aspects partially within our smart home case study (cf. Chapter 7), but more detailed experiments are required as part of possible future work. The application of stream-based process mining to check for process conformance at runtime (i. e., the correct process execution behaviour based on the predefined model) [JKM+17] is also a possible future extension to facilitate the self-management of workflows. As already discussed in Section 6.2.6, more sophisticated means of adapting the process instances–not only on the resources level but also on the structural level [DR09, MGR04], selecting dynamic subprocesses and services based on *Worklets* [ATHEVDA06] or synthesizing new compensation/remedy workflows [RSA10]–could be integrated as future improvements to achieve a higher degree of self-adaptation. Due to the relatively generic implementation of the MAPE-K control loop by the Feedback Service, we have a general framework for enabling self-management of workflows w. r. t. arbitrary criteria and metrics that can also be used in other non-workflow related contexts. As discussed in the previous section, *Consistency Levels* can be used to define the level of precision or fulfilment of the respective objectives that define success or error criteria for the self-management of the workflows and information systems via MAPE-K control loops.

6.7. Adaptations and Meta-adaptations

The implementation of the MAPE-K-based feedback loops in our approach follows the three layer architectural model for self-management by Kramer and Magee as shown in Figure 6.8 [KM07]. On the *Component Control* layer, processes and CPS entities provide status updates and other sensor-related information to the *Change Management*. Here, plans to react to the new situations are chosen and change actions are propagated to the component control. The upper *Goal Management* layer is responsible for deriving new change plans in accordance with the state updates, available change actions and higher level goals. These higher level goals correspond to the goal definitions for the workflow activities, the change actions correspond to the actions to be derived to compensate for the determined mismatches during process execution.

To make our implementation of the self-management capability based on the MAPE-K approach more flexible, also the selection/creation of goals and the selection/creation of compensation strategies could be enhanced by applying MAPE-K loops for the individual phases (*Multi-level Feedback Loops*). Currently, we rely on rather static goal definitions regarding static sensors and sensor values evaluated within static MAPE-K loops. Changing the sensors and thresholds used for defining the context paths, satisfied conditions and compensations at runtime based on additional information related to other context factors could further increase the flexibility and degree of self-adaptation of our approach (*Meta-adaptations* [PRK+14]). A possible way of implementing this approach based on our existing concepts is to also model the phases of MAPE-K loop–currently viewed as an internal and integrated process–as explicit process steps following the metamodel presented in Chapter 4 and adding goal definitions to the respective process steps that should be managed and adapted at runtime. This MAPE-K process would then be enacted by PRO-

tEUS and the Feedback Service would be responsible for executing the MAPE-K loops for the MAPE-K subprocess steps (Monitor, Analyse, Plan, Execute), e. g., to adapt the goal specifications of the *Analyse* process step. Extending the basic concepts with this approach and testing the adaptation also regarding these dynamic multi-level feedback loops could be part of future developments.

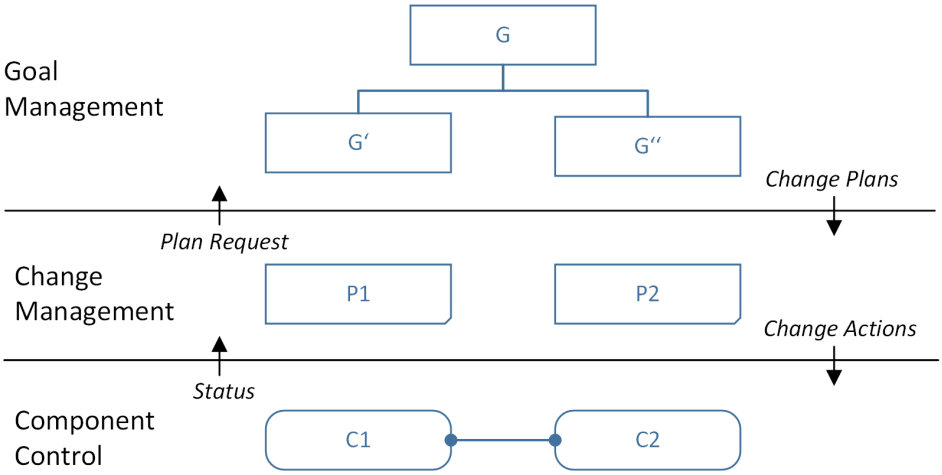


Figure 6.8.: Three Layer Architecture Model for Self-Management from [KM07].

6.8. Multiple Feedback Loops and Process Instances

As shown in the previous examples, there can be multiple instances of MAPE-K loops executed on the peer level (e. g., for ensuring Cyber-physical Consistency; cf. Section 6.3) and also on the super-peer level (e. g., for checking the liveliness signals of peers; cf. Section 6.4). With every objective defined for a process step and every instance of the process step to be executed, there is also an instance of the MAPE-K loop running in the Feedback Service on the super-peer or peer. Exemplary performance evaluations of the execution of single MAPE-K instances can be found in Chapter 7. A more extensive evaluation regarding a larger number of parallel feedback loops and with that potential performance issues is subject to future work.

In general, we assume that the process modellers are responsible for defining reasonable and non-conflicting processes and process goals. Consistency Style Sheets support the designers with finding the correct parameters at this point. However, this far, we mostly focussed on the modelling of individual CPS workflows and the execution of single instances of these workflows. The number of parallel process instances in CPS is likely to increase with the number of involved entities and increasing complexity of CPS. In contrast with "traditional" business processes that involve scalable virtual resources (software, services, memory, computations, etc.), processes in CPS also interact with physical resources that may not be scalable and thus result in conflicting accesses to these constraint resources. A partial remedy for this issue is to design the goals and objectives used in the feedback loops accordingly to define the local and global constraints for the process instances (e. g, the

maximum energy consumption for all process steps in a certain amount of time). This requires a global and shared knowledge base containing all relevant context factors and resource limitations. Due to the occurrence of emergent behaviour in CPS and especially CPSoS [CBF+16], it may not be possible to anticipate every possible situation and feasible to specify all possible constraints in the respective goals, though.

6.9. Transactions and ACID for CPS Workflows

The previous discussions and the detailed investigation of *Cyber-physical Consistency* for workflows (cf. Section 4.6) as an extension of the "classical" *Consistency* property for distributed systems [GR92] raise questions regarding the extensions of the other ACID criteria towards *cyber-physical ACID* and cyber-physical transactions for workflows in general. Here we briefly discuss how to possibly realize the individual criteria for CPS workflows and potential pitfalls on the conceptual level:

Atomicity Atomicity refers to the "all or nothing" nature of actions in distributed system or databases, i. e., the action is either completed successfully or in case of failure, the action has to be undone and the previous state restored. In CPS this requirement is harder to achieve as actions in the physical world cannot be undone that easily. The rollback and undo mechanisms are often far more complex in the physical world (e. g., repairing a broken window due to the service robot having crashed into it), which requires complex undo and compensation strategies or a high level of ensuring safety-critical behaviour to guarantee the successful execution or to restore the previous states.

Consistency The topic of keeping the workflow execution in the physical world and the cyber world in a consistent state has been discussed in detail in previous sections of this thesis (cf. Sections 4.6 and 6.3). In case of errors in the physical or virtual world, the state of the counterpart object, context factor or process has to be kept in sync–either by updating the state of the (virtual) counterpart or by fixing the error with respect to the original (physical) entity.

Isolation Isolation refers to the concurrent execution of actions and their result being the same as if they were performed in sequence. For CPS this property is again very hard to achieve as actions performed in the physical world have a stronger impact on the involved physical entities than in the virtual world. With many parallel process instances and other entities (e. g., humans) continuously influencing one or more physical objects, a software-based locking mechanism to handle concurrent access to physical resources and objects seems to be only a partial solution. Increasing the usability of the CPS control systems for users and notifying them about possible conflicting interactions among processes and their own actions may help to remedy this problem. A variety of mutually influencing context factors and processes have to be considered at this point.

Durability Durability is meant to ensure/persist the state of an action after its execution even in case of power outages or other unforeseen events. In CPS this

property is again hard to guarantee as the states of physical objects influenced by processes or other entities are harder to maintain. Many physical entities (e. g., humans or animals) are able to influence the resources affected by the execution of actions and processes/workflows. On the other hand, these physical entities cannot always be controlled by software and therefore, the state of a physical object after or during the process execution cannot always be guaranteed. The access to a cyber-physical object (cf. Section 6.10) cannot be completely locked with the help of software. A way to at least detect failures with respect to Durability is to constantly update the underlying world models including the states of the relevant objects with the help of sensors to keep the states of the physical and virtual world consistent.

Cyber-physical ACID

The findings show that ACID properties for transactions and workflows are harder to achieve and maintain in CPS due to the newly introduced physical dimension and interactions with the physical world. Corresponding goal specifications and executions of the MAPE-K loops may facilitate the maintenance of the ACID criteria (e. g., with respect to concurrent access or consistency). In [MMG08] Montagut et al. discuss the implementation of *Pervasive Workflows* that also support long running transactions. They basically define "critical" zones within processes and corresponding rollback processes and actions to be able to revert the execution in case of errors. These critical zones could also be part of our workflow definitions that can be considered within the *Analyse* and *Plan* phases of the feedback loop executions in future developments. The implementation of a *Two-phase Commit Protocol* [Lec09] for CPS could also be a good starting point for further discussions. Another approach to fulfil the ACID criteria to a certain degree could be to apply principles of role-based programming to the processes and entities involved in the execution in future work. Roles could be assigned to processes and cyber-physical objects/resources. Goals and objectives in combination with *Compartments* could then be used to specify allowed and forbidden concurrent interactions between multiple processes (instances) and physical entities depending on different contexts as well as consistency criteria to be maintained by the compartments [KLG+14]. Further more detailed discussions of applying these concepts in the context of cyber-physical workflows are required as part of future work.

6.10. Runtime View on Cyber-physical Synchronization for Workflows

Figure 6.9 shows our view on *Cyber-physical Synchronization* and *Cyber-physical Objects* at runtime illustrated with the help of the smart lighting example. Real world objects (e. g., a lamp) belong to the physical M_P0 layer in accordance with the MOF levels [Omg08]. These real world objects are represented by a corresponding virtual software object on the cyber M_C0 layer (*Digital Twin*). This object is an instance of an abstract model (class) on cyber level M_C1 describing the attributes and methods that instances of the class will have (here: class *Lamp*). The level of abstraction depends thereby on the requirements the corresponding software systems need to fulfil. A complete coverage of all the attributes and behaviours of the physical

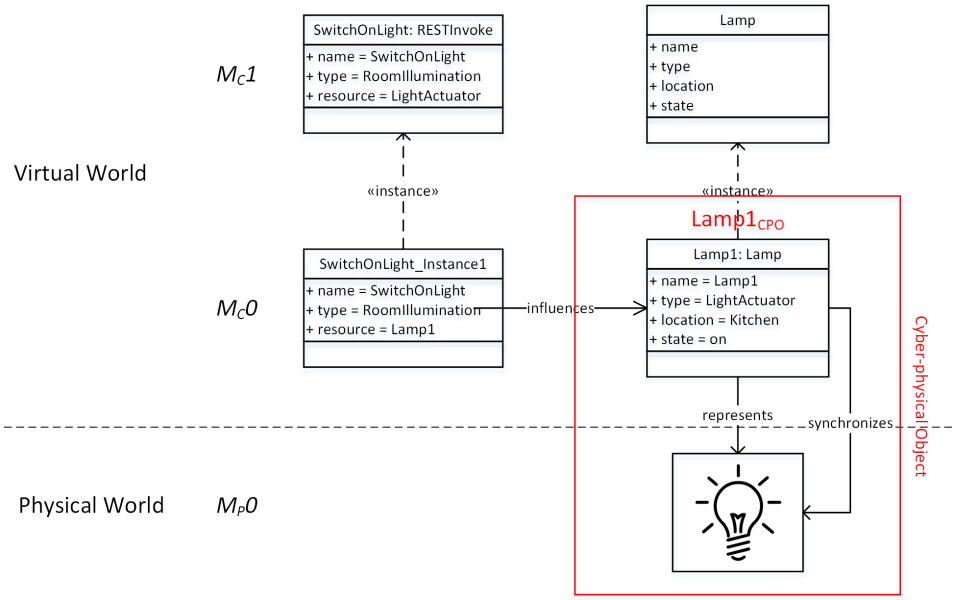


Figure 6.9.: Runtime View on Cyber-physical Objects and Synchronization Influenced by a CPS Workflow.

object that the virtual object represents is usually not feasible or even possible. The physical object (concrete physical lamp) on M_P0 and the representing cyber object (*Lamp1* object) on M_C0 form a *Cyber-physical Object* ($Lamp1_{CPO}$). This split-up of the cyber-physical object in a physical and a virtual part raises the question with respect to the existence and similar split-up of a *cyber-physical model* on level $M1$. The *Lamp* class on level M_C1 represents the abstract software (cyber) model of the concrete lamp "Lamp1". A more abstract physical model on level M_P1 could be a physical blueprint or template of the model of the lamp.

The cyber part of the cyber-physical object is a software artefact that is able to interact with its physical equivalent–either directly using the object's control software for sensing of states and actuating via control commands, or indirectly using services and external sensors or actuators. These internal or external services, sensors and applications are used for *Cyber-physical Synchronization*, i. e., keeping a consistent state of the physical object and its virtual representation in the sense of *Cyber-physical Consistency* (cf. Section 4.6). This can either be achieved by updating the cyber object's state after a change within the physical object's state as determined by corresponding built-in or external sensors, or by executing compensation actions to influence the physical object's state according to the cyber object's state. The selection of one of these ways depends on the assumed correctness of the physical or cyber object's state, which is defined within the process *goals* in our approach. For our smart lighting example described in Section 6.3, we assume that the cyber object's state is the correct/desired state, hence we influence the physical object's state accordingly. The other choice would be to update the virtual object's state to *Lamp1* being off and execute no further operations. However, the instance of the "SwitchOnLight" process that influences the virtual object's (*Lamp1*) state would then run into an error or inconsistency of process states.

As already pointed out in Section 4.6, there is no direct representation of a process instance in the physical world (*Physical Twin*). Its physical effects can only be observed within changes in the states of physical entities (objects, humans, etc.), the physical environment, or other context factors. Therefore, the cyber-physical synchronization aspect for workflows has to also cover the physical context factors regarding the cyber-physical *objects* and the environment.

6.11. Applicability of Workflow Feedback Loops to other CPS Domains

In Section 4.9 we elaborate on the compatibility of the proposed workflow notation with related business process languages and in Section 5.7 we discuss the suitability of the proposed WfMS for CPS to serve as reference architecture for other CPS domains. Thus far, the examples used to describe the concepts related to the self-management of CPS workflows refer to scenarios from the smart home domain. However, the goals used to define the outcome of the process activities can also be related to specific machine states, factory context values, geospatial locations or quality attributes of component parts or produced goods in the context of smart factories and associated supply chains. The examples from the logistics domain to monitor the transport of goods using GPS data as part of supply chain processes presented in [BCD+15] can also be investigated to be adapted within our feedback loops for workflows in future work. In general, the relevant context attributes analysed in the MAPE-K loops need to be measurable via the Feedback Service and integrated into the Knowledge Base. Goals can then be specified according to the criteria related to cyber-physical consistency, QoS, KPIs or other arbitrary factors related to various domains. The Compensation Repository has to contain the specific Compensation Queries and strategies with respect to the occurred mismatches. The compensation actions and the corresponding services to be invoked by the Executor also have to be stored in the Knowledge Base.

The application of the MAPE-K approach for workflows in other CPS domains (e. g., in smart hospitals, smart cities, smart factories or automotive) remains subject to future work. The concept of applying feedback loops to monitor, analyse and adapt software systems controlling physical actuators is widely accepted and used in practice on different levels of abstraction [GPGV14]. With relying on service-based communication and a decoupled standalone micro-service application for implementing the proposed feedback loops in form of the *Feedback Service*, we are currently limited to CPS domains with relatively low requirements regarding safety-critical and real-time demanding behaviour due to the computational and communications overhead introduced by the SOA-based approach. We assume that the basic mechanisms and processes related to the safety of the CPS devices and interactions are implemented on a level close to the respective hardware's control applications (cf. Section 2.5.1)–very likely also applying feedback loops involving sensors and actuators (e. g., the bumper sensor of the service robot indicates the need for changing its movement direction).

With goal definitions and applying the MAPE-K loop on the business process level and service-based interactions with sensors and actuators, we are able to support non-critical, mostly asynchronous and long running process executions and adapta-

tions, e. g., as shown with our case study in the smart home domain (cf. Chapter 7). More real-time and safety-critical feedback loops in CPS workflows (e. g., for verification and adaptation in the automotive domain [KGC+12]) have to be more tightly integrated into the corresponding control applications or WfMSes due to possible performance issues with a dedicated SOA-based software component. Goals can still be used to define expected and undesired outcomes for these self-managed systems/workflows [KM07]. A more detailed discussion with respect to the support of time-critical behaviour as well as safety and security-related aspects to be considered in feedback loops and supported by our approaches are given in Sections 7.6.9 and 7.6.10. An extended evaluation of the Feedback Service's performance in the case of executing multiple feedback loops for multiple process instances and an investigation of their mutual influences have to be part of future work when applying the workflow feedback loops to other CPS domains.

6.12. A Retrofitting Framework for Self-managed CPS WfMSes

In Section 3.2 a variety of existing WfMSes from industry and academia is presented briefly. Despite a wide variety BPM systems being already used in different contexts and domains, only a few systems possess capabilities that allow self-management to a certain degree. With requirement *R8*, we identified the need of adding these capabilities to existing WfMSes as there are already many systems deployed in industry and academia. In the following, we discuss different ways of *Retrofitting* existing *legacy* WfMSes with capabilities towards enabling the self-management of CPS workflows using MAPE-K loops implemented by the Feedback Service [SHA18a].

6.12.1. Retrofitting Process

The main purpose of most existing WfMSes is to enable service orchestration across (enterprise) applications and systems. The WfMSes investigated in Section 3.2 rely on SOA principles to find and invoke external or internal web services. With the Feedback Service as a dedicated web service providing a RESTful service interface to be invoked from within a process or another service, the integration with other WfMSes and existing worfklows is straightforward. The MAPE-K approach serves as basic retrofitting framework. We distinguish between *Invasive Retrofitting* and *Non-invasive Retrofitting* with and without the use of Consistency Style Sheets.

Invasive Retrofitting

The invasive way of retrofitting legacy WfMSes comprises modifications to the underlying workflow metamodel as well as to the internal execution processes of the WfMS. An example for the invasive retrofitting of the basic PROtEUS workflow system is shown in this thesis regarding the modelling of CPS workflows (cf. Section 4.5) and managed process steps in general (cf. Section 4.5.2), and regarding their execution (cf. Section 6.2). The basic process step classes of the workflow metamodel (activities, subprocesses, processes) have to be extended with new attributes or classes to specify goals and objectives as described in Section 4.5. The

corresponding workflow IDE should be adjusted accordingly to provide the designers with the possibility of modelling goals for specific workflow activities and process steps. When executing a (*managed* or *cyber-physical*) process activity augmented with a new goal, the workflow engine has to issue an additional parallel service call containing the relevant goals to the Feedback Service.

Figure 6.10 shows the process of executing a CPS activity by a retrofitted legacy WfMS. Upon executing an instance of the CPS activity or subprocess, its goal is extracted and used as parameter for an implicit parallel call to the Feedback Service. The Feedback Service then executes the MAPE-K feedback loops during the execution of the CPS activity to analyse context data with respect to the defined satisfied and compensation conditions, and to try to reach the goal. The service response including the state of the fulfilment of the goal is reported back to the WfMS. In case of errors or the Feedback Service not being able to fulfil the goal, the workflow system's internal error handling mechanism or associated process failure branches have to be activated.

Figure 6.11 shows the execution of a CPS activity by a retrofitted WfMS using a Consistency Style Sheet, which contains the goal definitions for the process (cf. Section 4.7). In this case, the particular Consistency Style Sheet is already known by the Feedback Service as it was submitted by the process designer to the service before. The implicit parallel request invoking the Feedback Service in parallel to the basic process step then only needs to contain the respective process step identifier. The Feedback Service links this identifier to the corresponding goal from the style sheet and executes the MAPE-K loops as described in Section 6.2 in parallel to the "basic" workflow activity. By using the style sheet mechanism, the "original" workflow needs to be only minimally modified by adding a flag that indicates if the Feedback Service should be called for self-management.

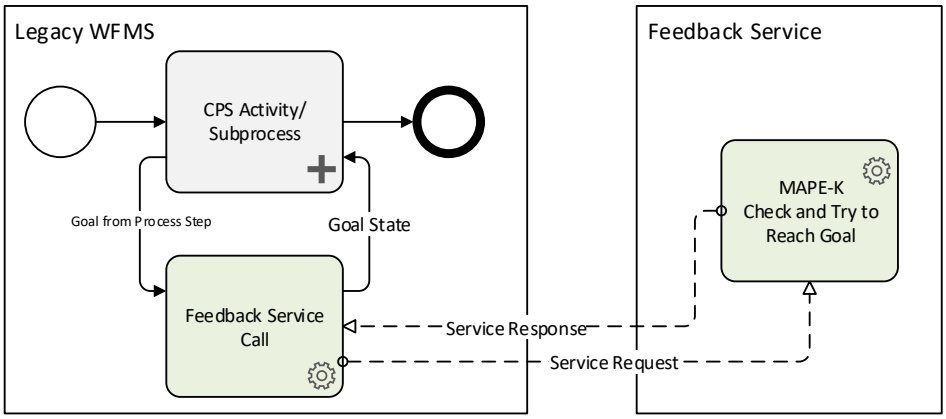


Figure 6.10.: Retrofitting Process for Existing WFMSes with Self-* Capabilities via the Feedback Service.

Non-invasive Retrofitting

The non-invasive way of retrofitting involves extending existing process models with explicitly modelled additional activities to call the Feedback Service. It does not require modifications to the workflow metamodels or engines–compared to the invasive

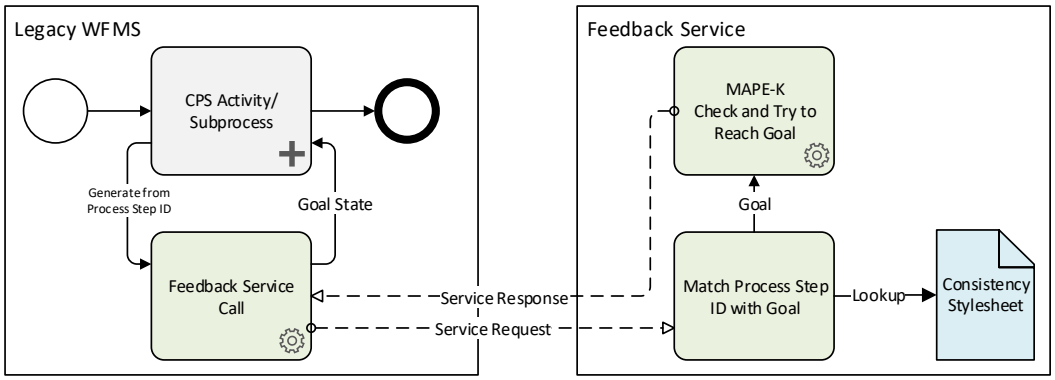


Figure 6.11.: Retrofitting Process for WFMSes with Consistency Style Sheets.

retrofitting. As the invocation of the Feedback Service is a simple additional service call, the WfMS's metamodel element for modelling a service invocation within a process and the WfMS's internal mechanism for calling the specified service can be used. The *Feedback Service Call* process activities shown in Figures 6.10 and 6.11 can be viewed as explicitly modelled process activities to be executed in parallel to the activity/subprocess to be managed. Goals and process instance information can be specified as input parameters for the service call to the Feedback Service. This service call then only has to be modelled as an additional process step to be executed in parallel to the "original" process activity that should be managed by the Feedback Service. The Consistency Style Sheets can be used similar to the invasive retrofitting as shown in Figure 6.11.

Invasive vs. Non-invasive Retrofitting

The invasive retrofitting process requires more modifications to the existing WfMS and its underlying metamodel. However, it provides a more intuitive integration into the existing WfMS and process landscape. Only a few new properties have to be added to the existing workflows. The workflow IDE can help to assist the process designer with specifying the goals of the managed process activities. As not all WfMSes are open source software or easily modifiable, this approach may not always be feasible to be implemented, though.

On the other hand, the non-invasive retrofitting approach requires more modifications to the existing process models and is less intuitive w. r. t. the modelling of a workflow as new process activities have to be added to the existing processes, which increases the complexity of the process models and possibly leads to concurring or blocking process executions due to the communication with the Feedback Service in parallel. However, the additional process steps and modified processes are still compatible with–but also limited to–the original WfMS and workflow language. Within the invasive retrofitting approach, modifications to the execution behaviours can be more diverse and less complex to realize in case adjustments or special process executions are required for specific use cases. Within the invasive approach, developers and process designers are limited to the expressiveness of the underlying workflow notation and to the capabilities of the corresponding WfMS.

6.12.2. Application to Existing WFMSes

The necessary steps for the invasive retrofitting process to add the capability of self-management to WfMSes are shown throughout this thesis–especially in Sections 4.5 and 6.2–for the basic *PROtEUS* WfMS. In this section, we briefly discuss the application of the *non-invasive* retrofitting approach to three of the existing open source WfMSes presented in Section 3.2. With *Activiti*, the *YAWL Engine* and *Apache ODE*, we have three complete WfMSes that use different underlying workflow languages, namely BPMN 2.0, YAWL and WS-BPEL (cf. Section 2.3.3). For all three WfMSes we show the retrofitting process of modifying the "basic" process with an additional service invocation containing the particular goals as parameters to call the Feedback Service in parallel to the invocation of the IoT (openHAB) middleware service for switching on the lamp in the *Morning Routine* scenario process. The results of the corresponding experiments using the three legacy WfMSes in combination with the Feedback Service in a non-invasive way and using PROtEUS with the Feedback Service in an invasive way to execute the Smart Lighting processes can be found in Section 7.5.

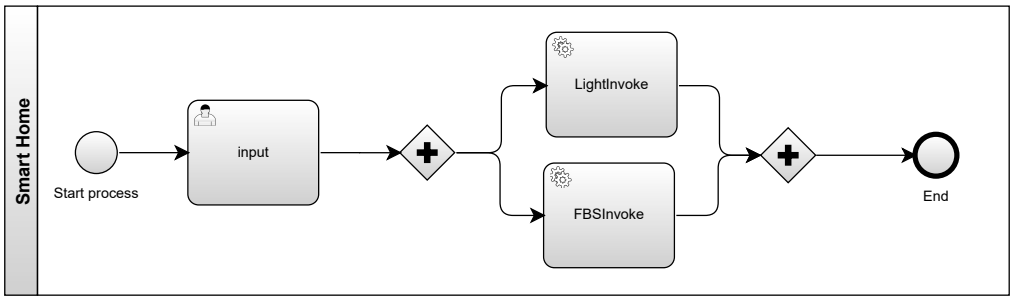


Figure 6.12.: Retrofitted Smart Lighting Process with Activiti.

Activiti Figure 6.12 shows the basic smart lighting process from our running example augmented with the additional call to the Feedback Service in BPMN 2.0. The *input* process step is used to provide input parameters. The basic process activity is the *LightInvoke* service task to call a custom service deployed on the Activiti system to trigger the dimmer switch via the openHAB middleware. In parallel, the invocation of the *FBSInvoke* service task calling the Feedback Service with the goal parameters to execute the MAPE-K feedback loops is specified. Activiti relies on intermediate services that are locally deployed for executing external functionality. In our example, these services are custom implementations that delegate the calls to the actual RESTful services provided by the IoT middleware and Feedback Service.

YAWL Engine Figure 6.13 shows the basic smart lighting process from our running example augmented with the additional call to the Feedback Service in the YAWL notation, which is based on Petri nets [vdAtH05]. The YAWL system also relies on custom services that are deployed in a local repository to execute functionality and invoke other external applications. We implemented custom services for each step of the process. The *input* and *output* services print input and output parameters for debugging purposes. The *LightInvoke* step is the basic process activity that calls the

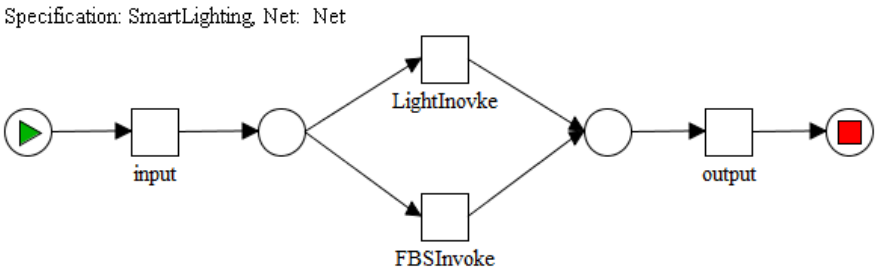


Figure 6.13.: Retrofitted Smart Lighting Process with YAWL.

openHAB middleware delegated via our custom service to activate the light dimmer. The *FBSInvoke* service is the "retrofitting" process step that invokes the Feedback Service delegated via our custom service in parallel. Goals are provided as input parameters for the *FBSInvoke* call. YAWL uses SOAP as internal communication protocol. We see that also with the YAWL system, additional custom services need to be implemented and deployed locally as intermediate services to call the external services. PROtEUS shows advantages with this regard as it as able to invoke external RESTful and SOAP-based web services directly.

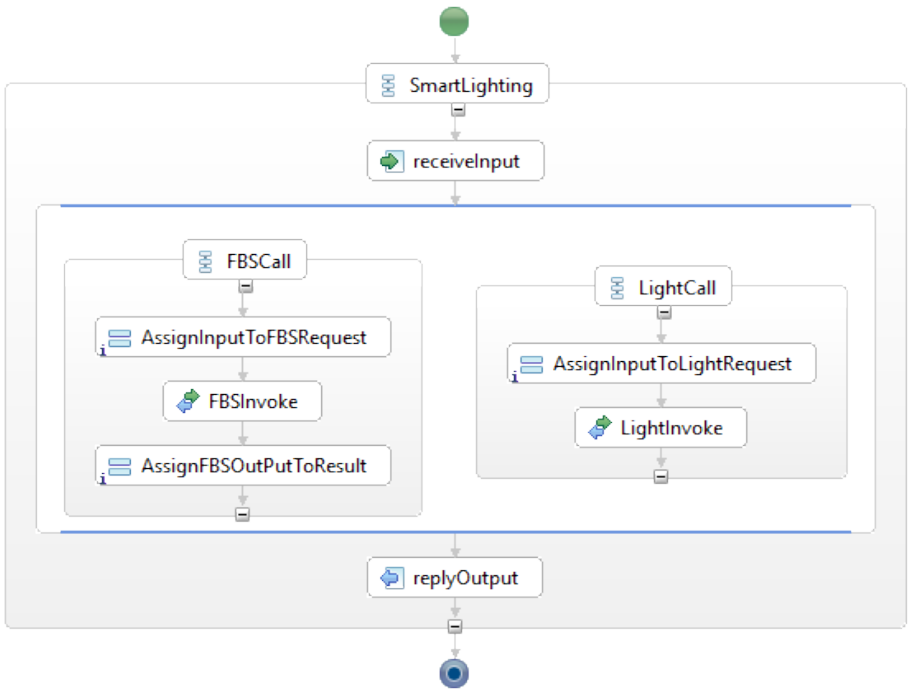


Figure 6.14.: Retrofitted Smart Lighting Process with Apache ODE.

Apache ODE Figure 6.14 shows the basic smart lighting process from our running example augmented with the additional call to the Feedback Service in graphical

WS-BPEL notation. The first process step is used to receive input data and assign it to the following service requests. The *LightCall* branch on the right side of the process contains the assignment of input parameters to the following basic *LightInvoke* step, which calls a custom service to then call the IoT middleware to switch on the light. In the parallel *FBSCall* branch, the *FBSInvoke* process step invokes a custom service to execute the MAPE-K loops via the Feedback Service with goals provided as input parameters from the previous process step. The replies are provided as output parameters. Similar to the previous examples, Apache ODE assumes SOAP-based communication with external services. It requires a WSDL description of the respective service requiring us to also provide intermediate services to delegate the service calls to the actual REST-based services of the middleware and Feedback Service.

6.12.3. Limitations

The examples for non-invasive retrofitting of the existing WfMSes show that usually only one additional process activity is required to add the capability of self-management based on the MAPE-K framework to individual workflow steps. However, the focus of most of the existing WfMSes is still on the SOAP-based invocation of custom services to implement a company's business processes and integrate enterprise applications. In contrast to that, the IoT and CPS increasingly rely on more light-weight service implementations and communication based on the REST protocol [GIM11]. This technology is also one the basic principles of the IoT middleware (cf. Section 5.3) and Feedback Service (cf. Section 6.2.1) to provide service based remote APIs. For that reason, we have to provide and deploy intermediate services to enable the communication between the SOAP-based WfMSes and the RESTful IoT services. A better support of direct REST service invocations would be a desirable feature for future developments of these WfMSes. The PROtEUS system already supports this kind of service calls. An alternative would be to add support for the more heavy-weight SOAP protocol to the existing IoT services and Feedback Service, which would probably be not always feasible due to resource constraints. In general, we assume that WfMSes are capable of service invocations and parameter passing to apply the MAPE-K framework for self-management with the help of Feedback Service, which is a dedicated standalone web service. In case the workflow system is not capable of establishing a service-based communication with the external Feedback Service or stricter performance requirements, the Feedback Service has to be integrated as additional software component and coupled more tightly to the basic WfMS as internal component. Running the Feedback Service as external micro-service introduces additional overhead due the hosting, operation and communication costs for the service.

Software or process engineers have to decide about which retrofitting process to choose based on the impact of the retrofitting process–either extending the meta-model and execution behaviour of the "basic" WfMS or extending the existing process models, which would maintain compatibility with the original WfMS. The selection of an appropriate "basic" WfMS depends on features and properties the WfMS has to provide and fulfil (e. g., formal verification or scalability) in the respective application domain or enterprise.

7. Evaluation

> "The true delight is in the finding out rather than in the knowing."
>
> Isaac Asimov

7.1. Introduction

In this chapter, we evaluate the new concepts presented in Chapters 4, 5 and 6 with respect to various criteria and the identified requirements. The basis for this evaluation are the two scenario processes introduced in Sections 2.2.1 and 2.2.2. We use the underlying processes or more complex subprocesses from these scenarios to provide a proof-of-concept evaluation of the new concepts regarding the modelling and resilient execution of self-managed CPS workflows. First, the basic *PROtEUS* system is used to execute the modelled *Morning Routine* and *Emergency* scenario processes. Afterwards, we show the application of the MAPE-K feedback loops using the *Feedback Service* to check for successful process execution and to deal with errors that have occurred. To show the feasibility of our concepts, we correlate the process instance executions with the corresponding changes in sensor and actuator states, and with the execution of the MAPE-K feedback loops. The experiments are partially based on the elaborations in [SHH17, HSKS16a, SHHA17, SKGA17, SHA18b]. In contrast to related approaches discussing their evaluation of CPS control and information systems based on just concepts or simulated data [MS17], we put our focus on conducting real world experiments to collect actual data from the physical world. Only with these kind of experiments, we are able to identify new unanticipated sources for errors and unexpected behaviour to then use the feedback loops to remedy situations and issues that occur due to the interactions with the physical world. Following the quantitative evaluation of our concepts in this smart home case study, we provide an extensive qualitative discussion regarding the fulfilment of the requirements *R1–R8* and additional aspects related to safety and security as well as questions that arose during the preceding elaborations. The results of these discussions show various advancements and improvements of our proposed concepts and prototypes over related approaches regarding the fulfilment of the individual requirements and also regarding the development of CPS WfMSes in general.

7.2. Hardware and Software

The following **Hardware** components were used to conduct the experiments in our smart home scenarios. Figure 7.1 shows the setup in our smart home lab.

Figure 7.2 shows the setup for a demo presented at the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp) [HSKS16a].

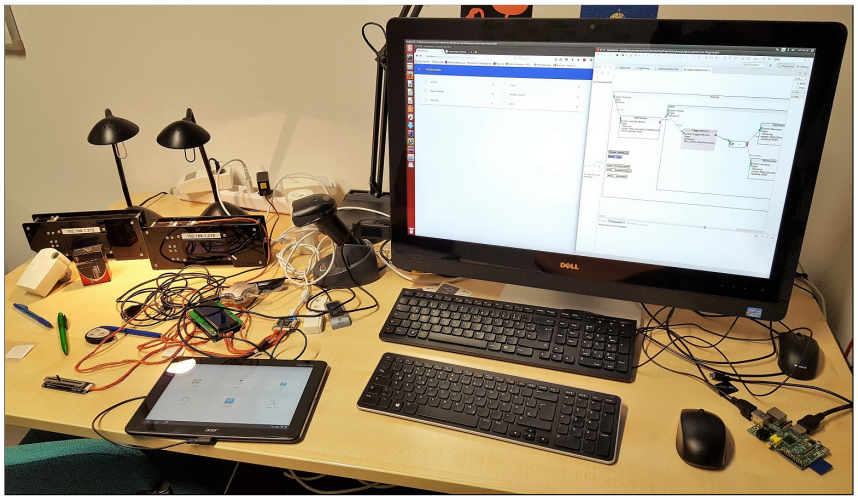


Figure 7.1.: Smart Home Lab Setup.

Figure 7.2.: UbiComp 2016 Demo Setup.

Sensors

- 2 Ambient light Bricklets and 1 infrared temperature Bricklet by Tinkerforge attached to Tinkerforge Master Bricks connected via USB
- 1 NFC reader connected via USB
- 1 BeSpoon tracking system with 3 tags and 1 anchor connected via USB

Actuators

- 2 Dimmer switches, 1 Gong, 1 KeyMatic door lock by Homematic connected to a HomeMatic CCU 1 via the BidCos radio protocol

- 2 LCD 20x4 Bricklets by Tinkerforge attached to Tinkerforge Master Bricks connected via USB
- 1 Smarter SMC10UK coffee maker connected via WiFi

Robots

- 2 TurtleBot 2 service robots with a Kobuki base station and an ASUS Xtion Pro 3D sensor
- Control laptop for TurtleBot 1: Asus X201E (Intel Celeron, 2 Cores at 1,1 GHz, 4 GB RAM, 100 GB HDD) running Ubuntu Linux 12.04 LTS (32 Bit) and ROS Groovy with ROSBridge 2[1] as WebSocket server, connected via WiFi
- Control laptop for second TurtleBot 2: Toshiba Satellite R630-14X (Intel Core i5, 2 Cores at 2,6 GHz, 8 GB RAM, 500 GB HDD) running Ubuntu Linux 12.04 LTS (64 Bit) and ROS Groovy with ROSBridge 2 as WebSocket server, connected via WiFi

Control Computer

- Dell XPS One 2710 (Intel Core i7, 4 Cores at 3,1 GHz, 8 GB RAM, 32 GB SSD, 2 TB HDD) running Ubuntu Linux 14.04 (64 Bit), connected via GB Ethernet

Miscellaneous

- ASUS AC200 RT as central network router with WiFi
- Acer Iconia Tab A510 Tablet with Android 4.1.2 for interactions with users

The following **Software** components were used to conduct the experiments:

- *openHAB 2.0*[2] running as Eclipse application on the control computer, with standard bindings for Tinkerforge, Homematic, HTTP and the Kodi media player, and with custom self-implemented bindings for the NFC reader, BeSpoon tracker and TurtleBots
- *PROtEUS*[3] WfMS running as Eclipse application on the control computer
- *Feedback Service*[4] running as standalone application on the control computer
- *SAL*[5] as plugin for openHAB
- *Neo4j* 2.3.2[6] as graph database running in a Docker[7] container on the control computer
- *ElasticSearch* 1.7.4[8] as logging and data analytics tool running in a Docker container on the control computer
- *Kibana* 4.1.5[9] as data exploration and visualization tool running in a Docker container on the control computer

[1] `http://wiki.ros.org/rosbridge_suite`
[2] `https://github.com/IoTUDresden/openhab-distro`
[3] `https://github.com/IoTUDresden/proteus`
[4] `https://github.com/IoTUDresden/feedback-service`
[5] `https://github.com/IoTUDresden/openhab2-addons`
[6] `https://neo4j.com`
[7] `https://www.docker.com`
[8] `https://www.elastic.co/de/products/elasticsearch`
[9] `https://www.elastic.co/de/products/kibana`

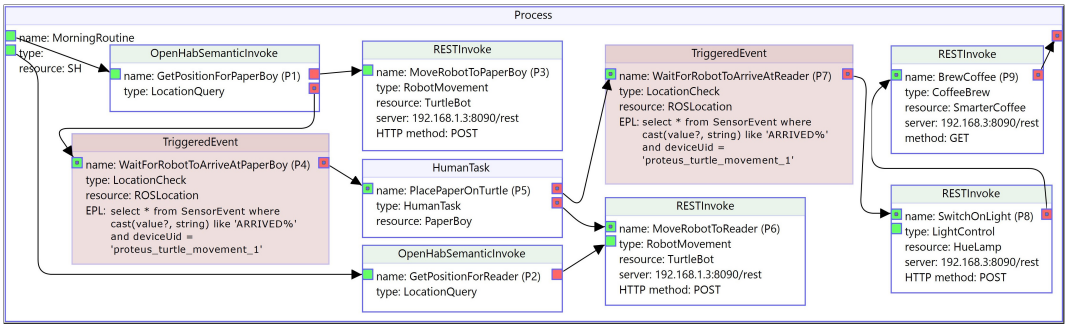


Figure 7.3.: Complete Morning Routine Scenario Process.

- *iBrew*[10] as Web service to interact with the Smarter SMC10UK coffee maker running on the control computer
- *DROiT API*[11] as Java-based programming abstraction for the TurtleBot robots
- *Smart CPS*[12] Android app for process and human task interactions on the tablet device and for simulation of health status data via a custom app
- *BeSpoonROS*[13] for transformation and publishing of BeSpoon position data via ROS running on the TurtleBot control laptops
- *Kodi*[14] media center for user notifications running on the control computer in version 16.1 and on the TurtleBot control laptops in version 14.2

7.3. PROtEUS Base System

The first two experiments relate to the processes described in Sections 2.2.1 and 2.2.2. With these processes providing smart home residents with assistance in an emergency situation and for comfort purposes, we will show the feasibility of the basic PROtEUS WfMS interacting with the openHAB IoT middleware, SAL [HSKS16b], service robots, other PROtEUS instances, and humans. These aspects address the identified requirements *R1–R4* (cf. Section 2.6).

7.3.1. Morning Routine Process

Figure 7.3 presents the complete *Morning Routine* process to be executed for this experiment in our workflow notation: a robot is sent to the paper delivery boy to get the paper; after the confirmation of the delivery the robot drives to the resident; upon the robot's arrival the light is switched on and the coffee maker starts brewing.

The first two process steps are used to get the position of the paper boy (*GetPositionForPaperBoy*, P1) and the reader (*GetPositionForReader*, P2) from the SAL. The *OpenHabSemanticInvoke* process class is a specialization of a *SemanticInvoke* process step type (cf. Section 4.4) allowing for dynamic service selection based on

[10] https://github.com/Tristan79/iBrew
[11] https://github.com/IoTUDresden/DROiTAPI
[12] https://github.com/IoTUDresden/smartcps
[13] https://github.com/IoTUDresden/BeSpoonROS
[14] https://kodi.tv/

Listing 7.1: SPARQL Semantic Select Query for Retrieving the Reader's Current Position.

```
SELECT ?position
WHERE {
        ?personClass rdfs:subClassOf* vicci:Person .
        ?person rdf:type ?personClass .
        ?person vicci:hasRobotPosition ?rP .
        ?person vicci:hasFirstname ?personName .
        ?rP vicci:hasOrientation ?o .
        ?rP vicci:hasPosition ?p .
        bind(concat('P:  ', str(?p), ' O: ', str(?o)) as
           ?position).
        FILTER(?personName = '<{ Reader}>' ) }
```

a SPARQL query. It contains additional parameters and a modified execution behaviour to use the SAL in combination with the openHAB middleware. The exemplary SPARQL query to retrieve the current position of the reader from the middleware and knowledge base can be found in Listing 7.1. The following REST service invocation (*MoveRobotToPaperBoy*, P3) uses the paper boy's current position as target coordinates for the robot to drive to. As this driving activity is an asynchronous action, the process engine waits for a specific *ARRIVED* event emitted from the robot to trigger the (*WaitForRobotToArriveAtPaperBoy*, P4) event in the process instance according to the corresponding EPL statement (cf. Figure 7.3). After the triggering of the event, the paper delivery boy receives a human task (*PlacePaperOnTurtle*, P5) to place the paper on the robot and confirm this action. Once confirmed, the process instance continues and the robot is sent via a REST service invocation to the reader (*MoveRobotToReader*, P6). Upon receiving the robot's arrival notification (*WaitForRobotToArriveAtReader*, P7) event, two subsequent REST service invocations are executed to switch on the light (*SwitchOnLight*, P8) and start brewing coffee (*BrewCoffee*, P9).

Experiments

We executed an instance of the *MorningRoutine* process as described above in the lab environment using the hardware and software components listed in Section 7.2. The positions of the reader and paper boy contained in the knowledge base are in the same coordinate system that the robots use. TurtleBot1 is the robot involved in this process starting from a random position. Figure 7.4 shows the robot's internal map of the room and the path it took to get from the the paper boy at the door to the reader (about 6 meters long). The Smart CPS app is used to present a human task dialogue to the paper boy to confirm the delivery of the paper to the robot. One of the dimmer switches turns on a desk lamp and the Smarter SMC10UK is used to brew the coffee. We used the *ElasticSearch* service to log the process execution data and changes within the states of the involved sensors and actuators.

Figure 7.4.: Robot's Path between Reader and Paper Boy for the Morning Routine Process (Black Lines = Obstacles/Walls, White Area = Discovered, Grey Area = Unknown).

Results

Table 7.1 shows the durations of the executions of the individual process step instances P1–P9 and the overall execution time of an instance of the *Morning Routine* process. A projection of these durations over time to show the flow of process step executions can be found in Figure 7.5. We summarized the execution times of certain process steps due the overall scale of the diagram and the large differences between the virtual and physical process activity executions regarding their durations.

The robot published an ARRIVED event approx. 23 seconds after starting to move to the paper boy, and approx. 51 seconds after starting to move to the reader, which resulted in the corresponding *TriggeredEvent* process steps (P4 and P7) to be executed. The corresponding asynchronous REST invocations calling the middleware to initiate the driving processes (P3 and P6) only took a few milliseconds. It took the paper delivery boy approx. 13 seconds to place the paper on the robot and confirm the human task on his tablet (P5). The *iBrew* web service responsible for communicating with the coffee maker responded after approx. 16 seconds to confirm the execution of the *Brew* command (P9). Compared to the short time of invoking the asynchronous web service for the light switch (P8), which is connected to the IoT middleware, the relatively long time of the iBrew service execution is due to the synchronous implementation and internal processes of the web service. The durations of the interactions with the SAL to retrieve the positions (P1 and P2) are also in the order of a few milliseconds.

Discussion

The results of the executions of an instance of the *Morning Routine* process using the PROtEUS WfMS show that the purely virtual process steps and computations are executed relatively fast–in the order of a few milliseconds, which indicates a good performance of the basic PROtEUS system. The interactions with the SAL for dynamic service discovery (Requirement *R2*), with the IoT middleware, robots, humans (Requirement *R3*), and external services work as expected. The event

Table 7.1.: Execution Times for Process Steps of the Morning Routine Process.

ID	Process Step	Duration (in ms)
P1	GetPositionForPaperBoy	34
P2	GetPositionForReader	42
P3	MoveRobotToPaperBoy	35
P4	WaitForRobotToArriveAtPaperBoy	23,631
P5	PlacePaperOnTurtle	13,218
P6	MoveRobotToReader	27
P7	WaitForRobotToArriveAtReader	50,859
P8	SwitchOnLight	13
P9	BrewCoffee	15,586
	MorningRoutine	103,334

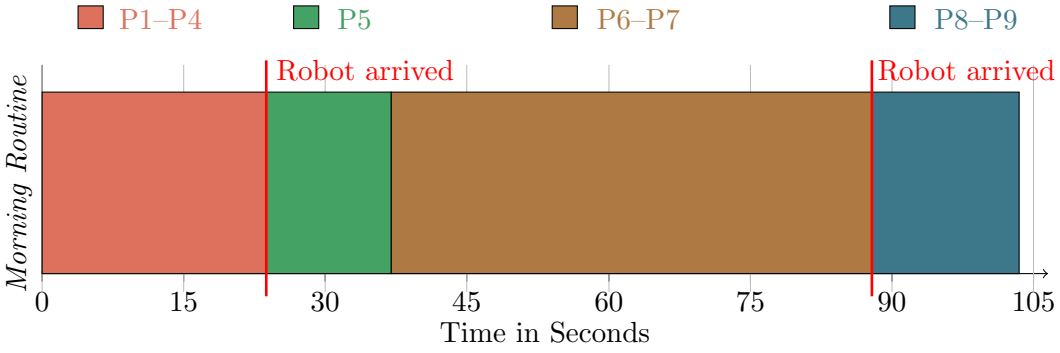


Figure 7.5.: Execution of the Process Step Instances of the Morning Routine Process over Time.

processing mechanism listening for the arrival events from the robot triggers the process-level events at the right moments in time, too (Requirement *R1*). The executions of the process steps interacting with the physical world take significantly longer than the purely virtual process steps, which is due to the mechanics and nature of the physical world leading to long running asynchronous physical process step executions. This shows the importance of supporting event-driven architectures and mechanisms to realize asynchronous processes within a CPS WfMS.

The results display the successful execution of an instance of the *Morning Routine* process. However, we had to restart the execution several times due to various errors related to the interactions with the the physical devices. Especially the internal autonomous navigation processes of the robot are very error-prone. The robot often gets stuck in narrow spaces with obstacles or it completely looses its internal position and cancels the whole process, which leads to the *Morning Routine* process to be cancelled or continued assuming the robot was successful (**Inconsistency**). Despite being stuck or not having arrived at the exact physical location, the robot often reports the successful execution back to the middleware leading to the emission of an ARRIVED event and an **inconsistent** cyber-physical state due to imprecisions of the robot's SLAM system. In this case, the WfMS assumes the correct final position

Listing 7.2: Semantic Command Query for Retrieving and Activating all TV-like Devices Capable of Playing Media.

```
SELECT ?func ?tv
WHERE {
 ?tvClass rdfs:subClassOf* dogont:Tv .
 ?playClass rdfs:subClassOf* dogont:PlayCommand .
 ?tv rdf:type ?tvClass .
 ?tv dogont:hasFunctionality ?func .
 ?func dogont:hasCommand ?cmd .
 ?cmd rdf:type ?playClass . }
```

of the robot and with that, the successful execution of the corresponding process step. The already described issues regarding the activation of the desk lamp–a broken or worn off light bulb–by the dimmer switch may also appear without the dimmer switch's internal control software or the WfMS noticing. In later experiments, we will show how to remedy these issues with the help of the *Feedback Service*.

7.3.2. Emergency Process

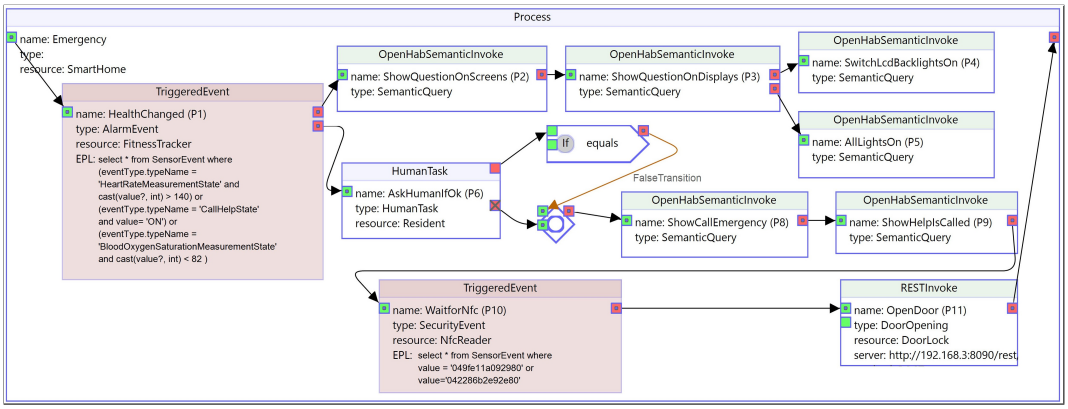


Figure 7.6.: Complete Emergency Scenario Process.

Figure 7.6 presents the complete *Emergency* scenario process to be executed for this experiment in our graphical workflow notation: after receiving data from a health monitor indicating a critical situation, all devices capable of displaying messages are activated to draw the resident's attention; in parallel, a human task is sent to the resident's tablet asking him/her to respond to the critical health situation; in case of a timeout or negative response, an emergency service is called; upon arrival and authentication of a medic, the door is opened automatically.

The first *TriggeredEvent* process step (*HealthChanged*, P1) uses the CEP engine to analyse data from a health monitor with respect to the EPL statement shown in the process model. If the heart rate is higher than 140 BPM or the oxygen saturation falls below 82 % or somebody already called for help, then this event is activated. Following, the human task (*AskHumanIfOk*, P6) is sent to the tablet device. In parallel, a semantic invoke process step (*ShowQuestionOnScreens*, P2)

Listing 7.3: Semantic Command Query for Retrieving and Activating all Displays.

```
SELECT ?func ?display
WHERE {
 ?displayClass rdfs:subClassOf* dogont:
    DisplayFunctionality .
 ?display dogont:hasFunctionality ?func .
 ?func rdf:type ?displayClass . }
```

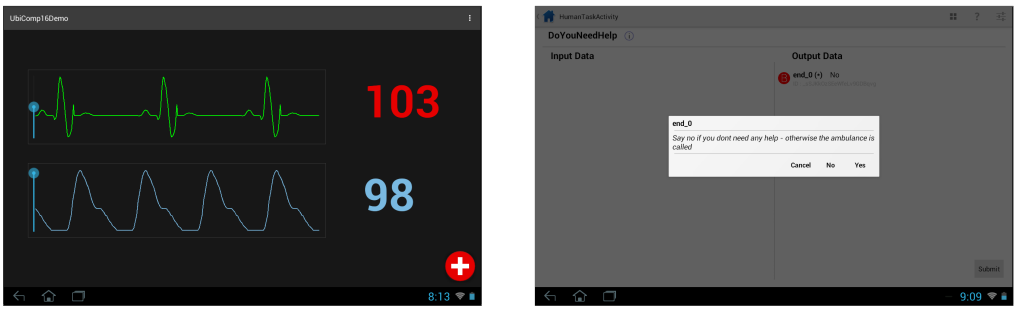
Listing 7.4: Semantic Command Query for Retrieving and Activating all Dimmer Switches.

```
SELECT ?func ?dimmer
WHERE {
 ?dimmerClass rdfs:subClassOf* dogont:DimmerSwitch .
 ?onClass rdfs:subClassOf* dogont:OnCommand .
 ?dimmer rdf:type ?dimmerClass .
 ?dimmer dogont:hasFunctionality ?func .
 ?func dogont:hasCommand ?cmd .
 ?cmd rdf:type ?onClass . }
```

is issued to display a message on all available screens (cf. SPARQL query in Listing 7.2), followed by a request (*ShowQuestionOnDisplays*, P3) to show a message on all displays (cf. SPARQL query in Listing 7.3), switch on the backlights of these displays (*SwitchLcdBacklightsOn*, P4), and switch on all available lights (cf. SPARQL query in Listing 7.4) to gain the resident's attention (*AllLightsOn*, P5). The human task process step contains an escalation port with a 15 seconds timeout. If this timeout is triggered or the resident's response regarding his/her health is negative, then an emergency service is called (*ShowCallEmergency*, P8) and a message is displayed on all screens (*ShowHelpIsCalled*, P9). The process engine then waits for an event triggered by the NFC reader (*WaitforNfc*, P10) in accordance with the EPL statement shown in Figure 7.6 listening for NFC tags with specific identifiers. Once the event is triggered, a REST service call is issued to the middleware to open the door (*OpenDoor*, P11). The process steps of type *OpenHabSemanticInvoke* are specializations of the *SemanticInvoke* process step type (cf. Section 4.4) allowing for dynamic service selection (*Semantic Command*) and invocation via the SAL in combination with openHAB.

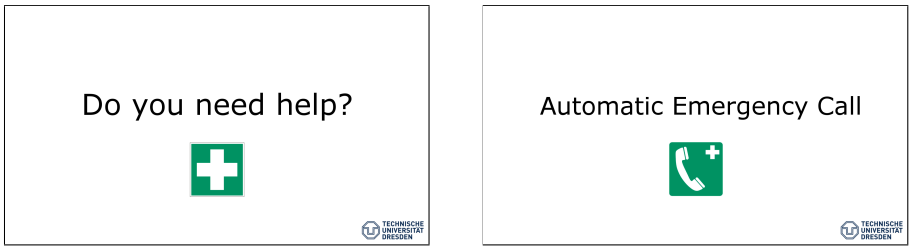
Experiments

We executed an instance of the *Emergency* process as described above in a lab environment using the hardware and software components listed in Section 7.2. A custom Android app is used to simulate the health monitor and publish corresponding vital signs to the openHAB middleware. A screenshot of this app allowing to adjust the heart rate and oxygen levels is shown in Figure 7.7(a). The Smart CPS app is used to interact with the resident via human tasks (cf. Figure 7.7(b)). The *Kodi* media center is used to display the messages "Do you need help?" (cf. Figure 7.8(a)) and "Automatic Emergency Call" (cf. Figure 7.8(b)) on the control computer and



(a) Health Monitor App. (b) Human Task in Smart CPS App.

Figure 7.7.: Android Apps used in Emergency Process.



(a) Help Message. (b) Emergency Service Call.

Figure 7.8.: Emergency Messages Displayed using the Kodi Media Center.

TurtleBot laptops. These messages are also displayed as text on the LCD units of the TinkerForge devices. The NFC reader is used for detection and authentication of a person at the door by specific NFC tags. The two HomeMatic dimmer switches activate two lamps and the KeyMatic actuator opens the door automatically. All devices and apps are connected to openHAB to publish events or trigger actions.

Results

Table 7.2 shows the durations of the executions of the individual process step instances P1–11 and the overall execution time of an instance of the *Emergency* process. A projection of these durations over time to show the flow of process step executions can be found in Figure 7.9. We summarized the execution times of certain process steps due the overall scale of the diagram and the large differences between the virtual and physical process executions regarding their durations.

Using the Android app, we let the heart rate and oxygen levels drop below the defined threshold after approx. 28 seconds (P1) (cf. Figure 7.10). This triggered the semantic invocations to display the messages on all available screens (P2) and displays (P3), and to switch on the available displays (P4) and dimmers (P5). The execution times of these semantic invocations range from 47 to 181 milliseconds depending on the complexity of the semantic query and ontology, and on the number of available instances [HSKS16b]. For this run of the experiment, we decided to not answer the human task (P6) and let the timeout trigger the following process steps after the defined timeframe of 15 seconds. The execution time of 7 milliseconds for

Table 7.2.: Execution Times for Process Steps of the Emergency Process.

ID	Process Step	Duration (in ms)
P1	HealthChanged	27,916
P2	ShowQuestionOnScreens	181
P3	ShowQuestionOnDisplays	47
P4	SwitchLcdBacklightsOn	57
P5	AllLightsOn	111
P6	AskIfHumanOk	15,046
P7	OR	7
P8	ShowCallEmergency	131
P9	ShowHelpIsCalled	55
P10	WaitForNfc	21,213
P11	OpenDoor	20
	Emergency	64,389

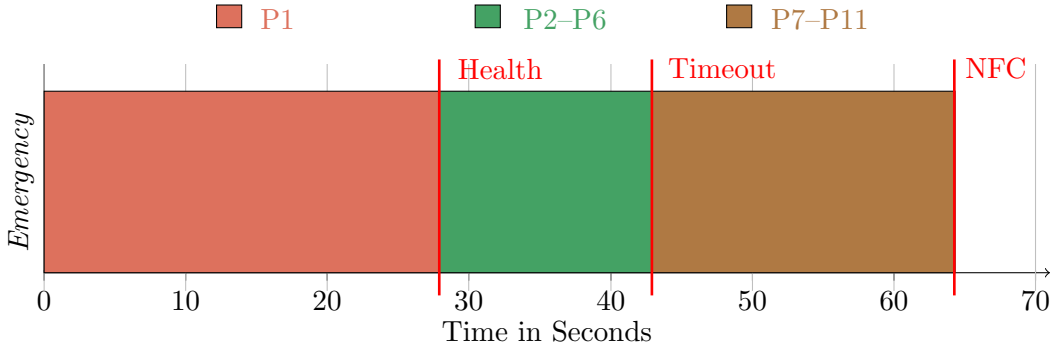


Figure 7.9.: Execution of the Process Step Instances of the Emergency Process over Time.

the subsequent *OR* join shows that PROtEUS also performs well executing basic logic functions. Following, two semantic invocations are executed to call the emergency service (P8) and display this message (P9). Both calls to the SAL match the execution times of previous interactions with the SAL. We waited approx. 21 seconds to activate the NFC reader with a correct NFC tag (P10), which is followed by a REST call to the middleware to open the door.

Discussion

For this scenario applying the basic *PROtEUS* WfMS to execute the process instance, we focussed on using the SAL at various points in time to dynamically find and invoke process resources (Requirement *R2*). Without prior knowledge of actual instances of available smart home devices, we can use the SPARQL queries to specify required functionality, device classes and other context constraints in single process steps that will trigger multiple devices at the same time (e. g., *all* available dimmer switches or *all* available displays). The results show that these interactions work as expected and with acceptable execution times of 47 to 181 milliseconds for

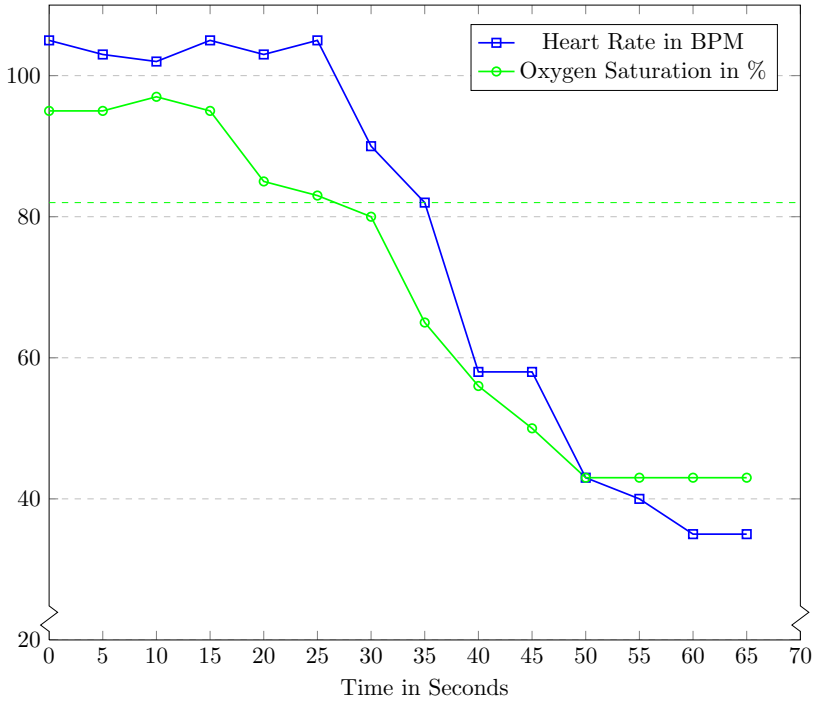


Figure 7.10.: Simulated Data of the Health Monitor.

the smart home. However, we also see slightly longer execution times compared to basic REST calls to the openHAB middleware. The results also show that the interactions among PROtEUS, SAL, middleware and humans (Requirement *R3*) have been executed successfully and the timeout mechanism as well as the CEP engine worked as expected (Requirement *R1*). Within this scenario, we can also find new crucial error sources regarding the interactions with the physical world (e. g., with respect to the light switches and the door opener).

7.4. PROtEUS with Feedback Service

With the previous examples, we showed the feasibility of the basic PROtEUS WfMS for executing complex CPS workflows, including the interactions with external services, sensors, actuators and humans. During the experiments, we encountered various types of errors regarding the cyber-physical actuators that cannot be handled by the basic PROtEUS system in a feasible way. The basic process descriptions need to be modified to remedy these new errors leading to much more complex process descriptions and additional process steps to handle all possible errors (e. g., with respect to the robot navigation or the light control). In the following experiments, we will use the *Feedback Service* [SHHA16, SHHA17] as additional component to try to detect errors and cyber-physical inconsistencies and also to automatically repair these errors by means of self-healing (Requirements *R5*–*R7*). We will also show examples of the distributed process execution using multiple instances of PROtEUS and D-PROtEUS in a peer–super-peer configuration (Requirement *R4*) supervised

Listing 7.5: Goal and Objective for MakeCoffee Process Step.

```
"MakeCoffee" : {
"name":"Coffee is ready",
"objectives":[
 { "name":"coffee temperature > 37 degrees within 3
    minutes",
   "satisfiedCondition":"#coffeeTemp > 37",
   "compensationCondition":"#objective.created.isBefore(#
      now.minusSeconds(180))",
   "contextPaths":[
    "MATCH (ctemp {name: '
       State_tinkerforge_irTemp_irTemp_1'})-[:
       hasStateValue]->(value)",
    "WHERE toFloat(value.realStateValue) > 0",
    "RETURN toFloat(value.realStateValue) AS coffeeTemp,
       id(ctemp) AS stateId"
] } ] }
```

by the Feedback Service [SHA17]. The coffee maker, the light dimmers and the service robots will be used as main cyber-physical actuators within these experiments.

7.4.1. Coffee Maker Verification

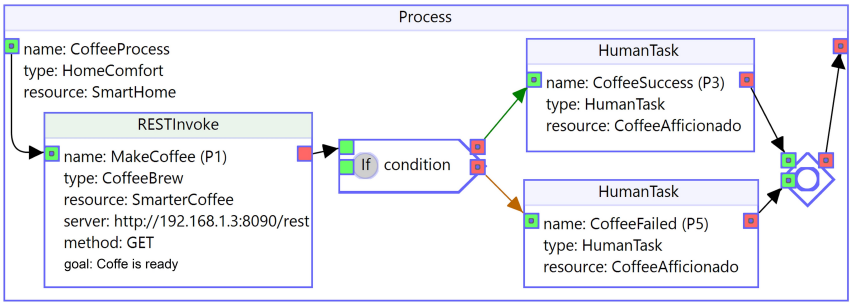


Figure 7.11.: Coffee Brewing Process.

For this experiment, we use the Feedback Service as additional software component involved in the process execution to verify the success of a *cyber-physical* process step: the brewing of coffee using the Smarter SMC10UK[15] coffee maker. Figure 7.11 shows the standalone *CoffeeProcess* as an excerpt of the *MorningRoutine* process. The *MakeCoffee* (P1) step is a REST service call to the *iBrew* web service enabling the remote communication with the coffee maker. This process step is marked as "cyber-physical" and contains the goal *Coffee is ready* presented in Listing 7.5. In order to verify the successful execution, an additional infrared temperature sensor mounted on the front of the coffee maker (cf. Figure 7.12) is used. This setup shows the extension of existing devices/machines with new additional sensors to increase the "smartness" of the respective device (*Retrofitting*). The objective states that this

[15] https://smarter.am/

new sensor (*tinkerforge_irTemp_irTemp_1*) should report a temperature of over 37°C within 180 seconds, which indicates that there is hot coffee in the cup. If this goal can be fulfilled (*satisfied condition* in Line 5), then a human task (*CoffeeSuccess*, P3) is executed notifying the user about the success. In case the temperature cannot be reached within the defined time frame (*compensation condition* in Line 6), a human task (*CoffeeFailed*, P5) is used to notify the user about a potential error. For this experiment, we do not try to find a compensation with the help of the Feedback Service but only to confirm the successful brewing or to detect errors within the process instance execution.

Figure 7.12.: *Retrofitted* Coffee Maker with Additional Temperature Sensor.

Experiments

We executed one instance of the *CoffeeProcess* using the PROtEUS WfMS and the Feedback Service to verify the execution of the service call to the coffee maker via the iBrew web service. The TinkerForge infrared temperature sensor is used to publish the additional temperature data to the IoT middleware and to the knowledge base. We prepared the coffee maker to execute a successful brewing process for this run.

Results

Table 7.3 shows the durations of the executions of the individual process step instances P1–P4 and the overall execution time of an instance of the *Coffee* process. A projection of these durations over time to show the flow of process step executions can be found in Figure 7.13. It took the *MakeCoffee* process step approx. 115 seconds to complete (P1). As shown in Figure 7.13, this step consists of the basic REST service invocation taking about 3 seconds to be enacted, extended by the execution

Table 7.3.: Execution Times for Process Steps of the Coffee Process.

ID	Process Step	Duration (in ms)
P1	MakeCoffee	114,940
P2	IF	38
P3	CoffeeSuccess	35,645
P4	OR	9
	CoffeeProcess	150,632

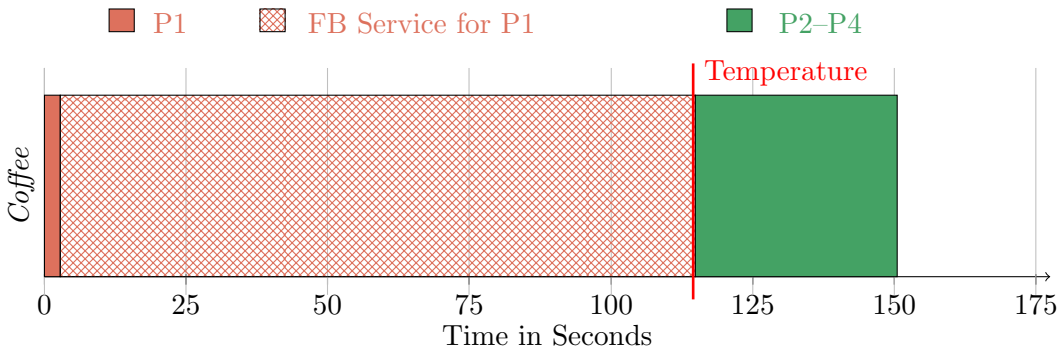


Figure 7.13.: Execution of the Process Steps of the Coffee Process over Time.

of the Feedback Service (FB Service for P1). The durations of the MAPE-K phases and the overall Feedback Service execution as well as the number of iterations can be found in Table 7.4. The corresponding values of the infrared temperature sensor over time are shown in Figure 7.15. We started the process with an initial temperature value of 28°C for the coffee cup. The threshold of 37°C defined in the satisfied condition was reached approx. 112 seconds after the Feedback Service was called to execute the MAPE-K loops for the instance of the "MakeCoffee" process step. Within this timespan, the Feedback Service executed 148 iterations of the MAPE-K loop taking on average 1023 milliseconds. The Monitoring (M) of the relevant context attributes (here: infrared temperature) is done continuously. The Analyser received 148 symptoms from the Monitor to evaluate new data. On average, the Analysis (A) phase with respect to the compensation and satisfied conditions took 6 milliseconds to be executed. As the satisfied condition was fulfilled within the defined maximum timeframe of 180 seconds, there was no need to initiate the Plan (P) or Execute (E) phases. The implementation and internal processes of the Feedback Service rely heavily on asynchronous communication, event driven behaviour and parallel threads. For that reason, a simple summation of the durations of the individual phases does not lead to the total execution times for the iteration of a MAPE-K loop or the entire Feedback Service. We inserted aspects regarding the logging of timestamps and publishing these data to the Elastic Search service at the relevant points within the source code of the Feedback Service to gather this monitoring data.

After the successful execution of the "MakeCoffee" process step (P1) and the evaluation of its outcome within the "IF" step (P2), a human task is sent to notify

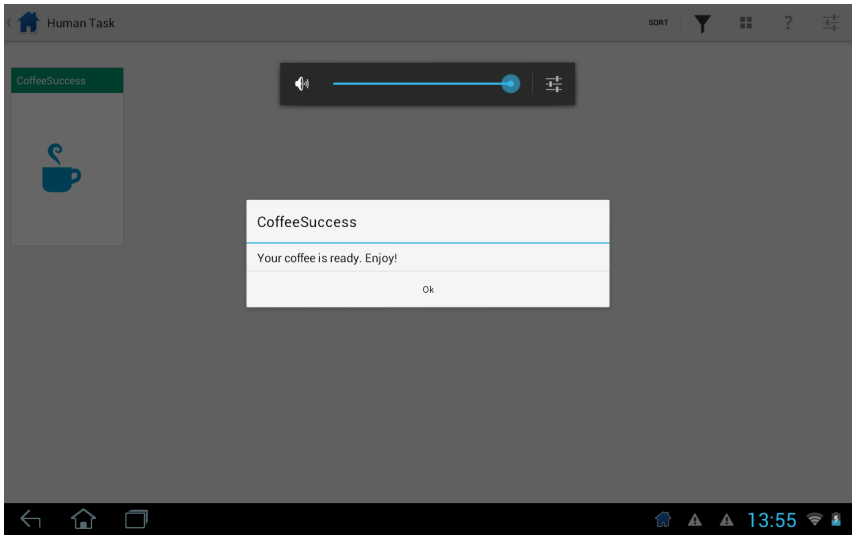


Figure 7.14.: Human Task for Successful Coffee Brewing.

Table 7.4.: Number of Iterations and Durations of the Individual MAPE Phases and Feedback Service for Process Step P1 of the CoffeeProcess.

Metrics \ Phase	M	A	P	E	Loop	FB Service
Iterations (#)	–	148	–	–	148	1
Duration (ø in ms)	–	6	–	–	1,023	112,135

the user about the successful coffee brewing (cf. Figure 7.14). It took the user approx. 36 seconds to confirm this task (P3) and with that, to terminate the process instance after the merge of control flow (P4).

Discussion

The results show the application of the Feedback Service to verify the successful execution of a *cyber-physical* process step based on information from an additional sensor. Using an infrared sensor to detect a change of the surface temperature of the coffee cup and with that, to derive new knowledge about the state of the coffee brewing process is one example of equipping existing devices or machines with new sensors (*Retrofitting*). We use this data and the specified goal to verify the successful execution of workflow activities including *Cyber-physical Consistency* of the CPS workflow instance after process execution (Requirement *R5*). The measured execution times for the Feedback Service components show an acceptable performance of the Feedback Service for the smart home context. The actual computations are in the order of a few milliseconds. The arrival of new sensor values and interactions with the physical world in general are the major influence factors for the overall execution times.

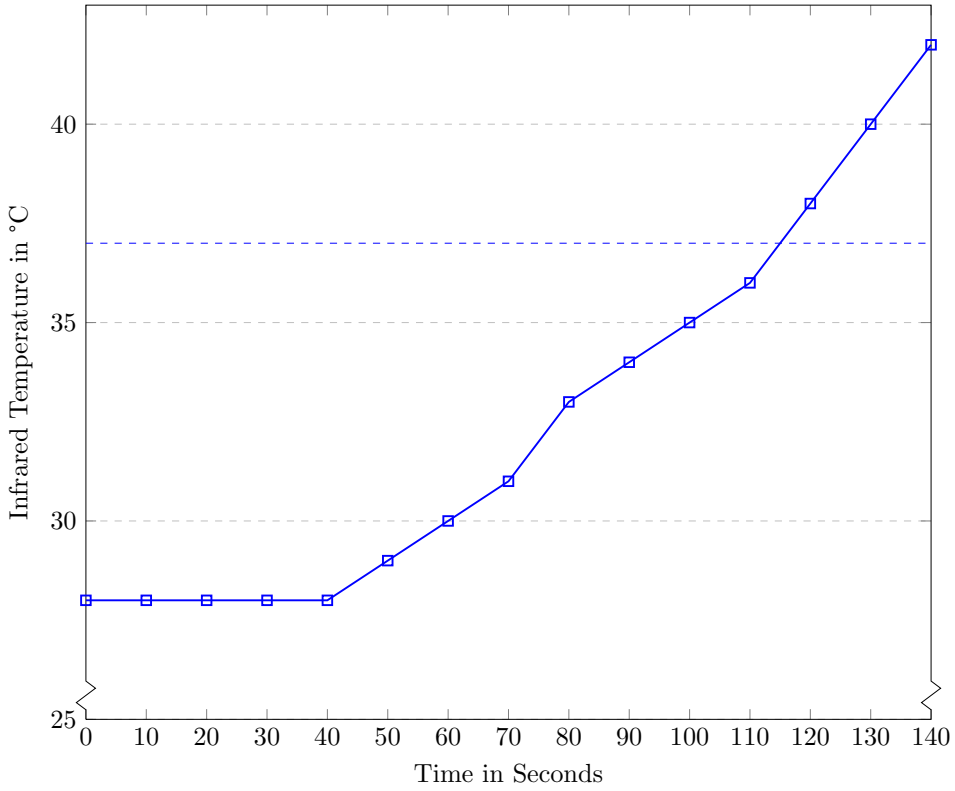


Figure 7.15.: Values from the Infrared Temperature Sensor over Time.

7.4.2. Continuous Light Control

This experiment refers to the automated light control within the *Morning Routine* and *Emergency* scenario processes [SHHA17]. We use the Feedback Service as additional component to verify the successful execution, detect inconsistencies and errors, and to repair these errors. Figure 7.16 shows the complete *LightControl* process. First, a specific dimmer switch is triggered to switch on the light (*SwitchOnLight*), then a loop (*LightControlLoop*) is used to enable a continuous control of the light levels. The overall goal is to ensure optimal lighting levels between 650 and 750 Lux in an energy efficient way by dimming the light switch up and down. In case of unresolvable errors, the user should be notified via a human task (*LightError*).

The *LightChange* event triggers the subsequent process steps according to the EPL statement to either dim up (*IncreaseLight*) or dim down (*DecreaseLight*) the light by increasing or decreasing the dimmer switch power levels by 10 % via REST service calls to the middleware. The *LightChange* event is triggered if the current light levels are below 650 Lux or above 750 Lux. We run an instance of the *LightControl* process in three configurations: *Baseline*, *MAPE-K+* and *MAPE-K++*.

Experiments

The PROtEUS WfMS and the Feedback Service are used to execute the process instances. A TinkerForge light sensor is used to measure the light levels and two HomeMatic dimmer switches are used to switch individual desk lamps on or off and

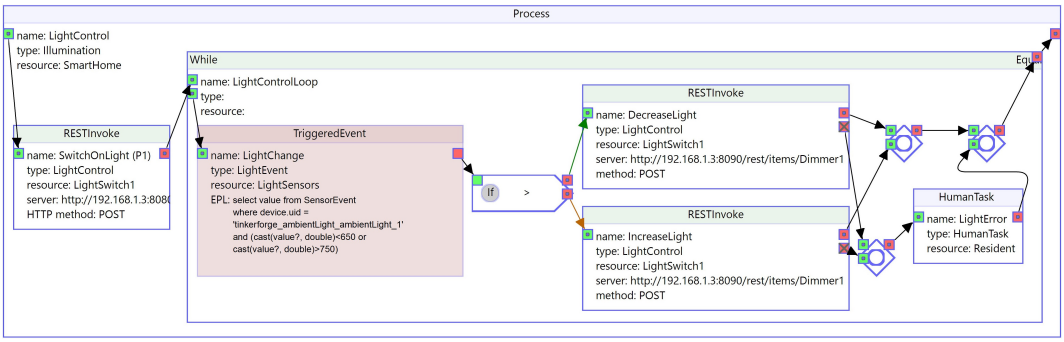


Figure 7.16.: Smart Lighting Process.

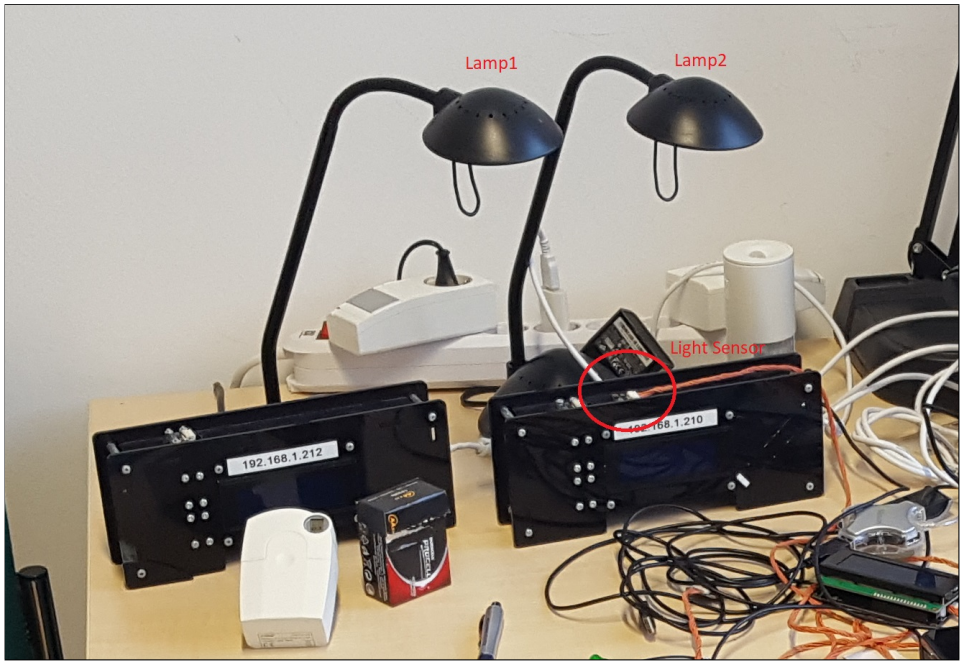


Figure 7.17.: Lab Setup for the Continuous Light Control Experiments.

to increase or decrease their power levels (cf. Figure 7.17). The sensors and actuators are controlled via the openHAB middleware.

Baseline The *Baseline* configuration executes the *LightControl* process without *cyber-physical* process steps (IncreaseLight, DecreaseLight)–thus without the Feedback Service–and without human task. Once a balance within the light levels is reached–between 650 and 750 Lux–and the control loop will not trigger new actions, we will turn of LightSwitch1 to simulate a burnt light bulb. The control loop will then continue to send the *Increase* command to *LightSwitch1* indefinitely as the light switch and the WfMS are not able to detect or fix the error.

MAPE-K+ The *MAPE-K+* configuration views the *IncreaseLight* and *DecreaseLight* process steps as *cyber-physical* process activities. Listing 7.6 shows the goal

Listing 7.6: Goal and Objective for IncreaseLight Process Step.

```
1 "IncreaseLight" : {
2 "name": "enough light for working",
3 "objectives":[
4  { "name": "kitchen light intensity > 700 Lux within 5
      seconds"
5    "satisfiedCondition":"#lightIntensity > 700",
6    "compensationCondition":"#objective.created.isBefore(#
       now.minusSeconds(5))",
7    "contextPaths":[
8     "MATCH (thing)-[:type]->(sensor {name: 'LightSensor
        '})",
9     "MATCH (thing)-[:isIn]->(room {name: 'Kitchen_Mueller
        '})",
10    "MATCH (thing)-[:hasState]->(state:
        LightIntensityState)",
11    "MATCH (state)-[:hasStateValue]->(value)",
12    "WHERE toFloat(value.realStateValue) > 0",
13    "RETURN avg(toFloat(value.realStateValue)) AS
        lightIntensity, head(collect(id(state))) AS
        stateId"
14 ] } ] }
```

property of the *IncreaseLight* process step. The execution of the process step has to lead to light levels above 700 Lux within 5 seconds to fulfil this goal (satisfied condition in Line 5). Otherwise, the Feedback Service has to search for a compensation action (compensation condition in Line 6). The respective light sensor values are referred to by the context paths (Lines 7–13). In analogy to the *Baseline* experiment, after a balance is reached by executing the light control loops, we switch off the first desk lamp, which will now trigger the Feedback Service to try to remedy this error. For this run, we made sure that the second dimmer switch is available and set up so that it is able to fulfil the goal. The goal for the *DecreaseLight* process step defines a satisfied condition of reaching light levels below 750 Lux within 5 seconds.

MAPE-K++ The *MAPE-K++* configuration corresponds to the MAPE-K+ setup. However, for this configuration, the second dimmer switch will also fail and no other compensating devices will be available, which will result in the Feedback Service not being able fulfil the goal. After the control loop reaches a balance, we switch off the first desk lamp. Once the Feedback Service restored the desired light levels with the help of the second dimmer switch by executing the increase and decrease operations, we also turn off the second desk lamp. This triggers the Feedback Service again as part of the next light control loop iteration. At this point, the Feedback Service is not able to find any other compensations from the knowledge base. The Feedback Service reports this error to the PROtEUS system, which leads to the execution of the human task "LightError" after activating the *IncreaseLight* process step's failure branch to notify the user.

Results

Table 7.5.: Execution Times for Baseline Light Control Process.

ID	Process Step	Duration (in ms)
P1	SwitchOnLight	353
P2	DecreaseLoop	1,759
P3	DecreaseLoop	1,352
P5	DecreaseLoop	611
P6	DecreaseLoop	1,609
P7	DecreaseLoop	340
P8	IncreaseLoop	1,182
P9	DecreaseLoop	1,786
P10	IncreaseLoop	52,026
P11	IncreaseLoop	14,278
P12	IncreaseLoop	998
	LightControl	76,294

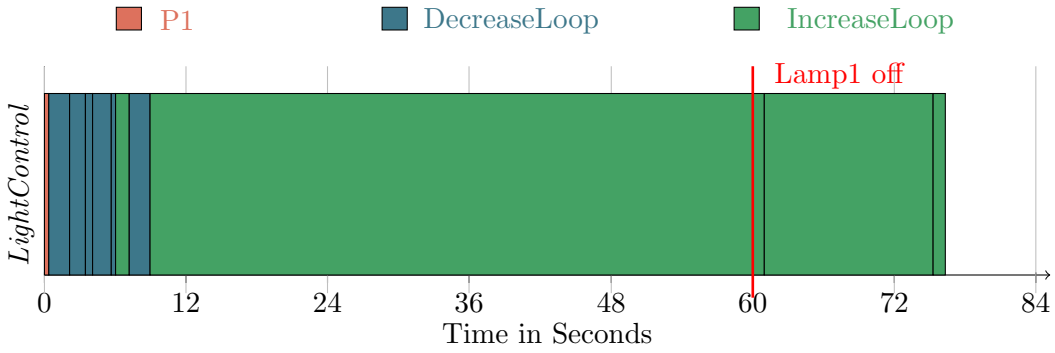


Figure 7.18.: Execution of the Process Step Instances of the Baseline Light Control Process over Time.

Baseline Table 7.5 shows the durations of the individual process step executions. We summarized the four process steps *LightChange*, *IF*, *DecreaseLight* or *IncreaseLight* and the following *OR* join as *DecreaseLoop* or *IncreaseLoop* respectively, i. e., one loop iteration consists of waiting for the LightChange event to be triggered, its evaluation to decide if the new light levels are too high or too low, and the service invocation to dim up or dim down the lamp. Figure 7.18 shows the flow of process step instance executions over time and Figure 7.19 presents the corresponding values of the light sensor and power levels of the dimmer switch. Following the initial activation of LightSwitch1 to 100 % power (P1), it takes several *DecreaseLoop* (P2–P7, P9) and *IncreaseLoop* (P8) executions to reach stable light levels between 650 and 750 Lux after approx. 10 seconds, which will not trigger the *LightChange* event anymore. Approximately 60 seconds after the start of the process execution (or 50 seconds after reaching the optimal lighting levels), we switch

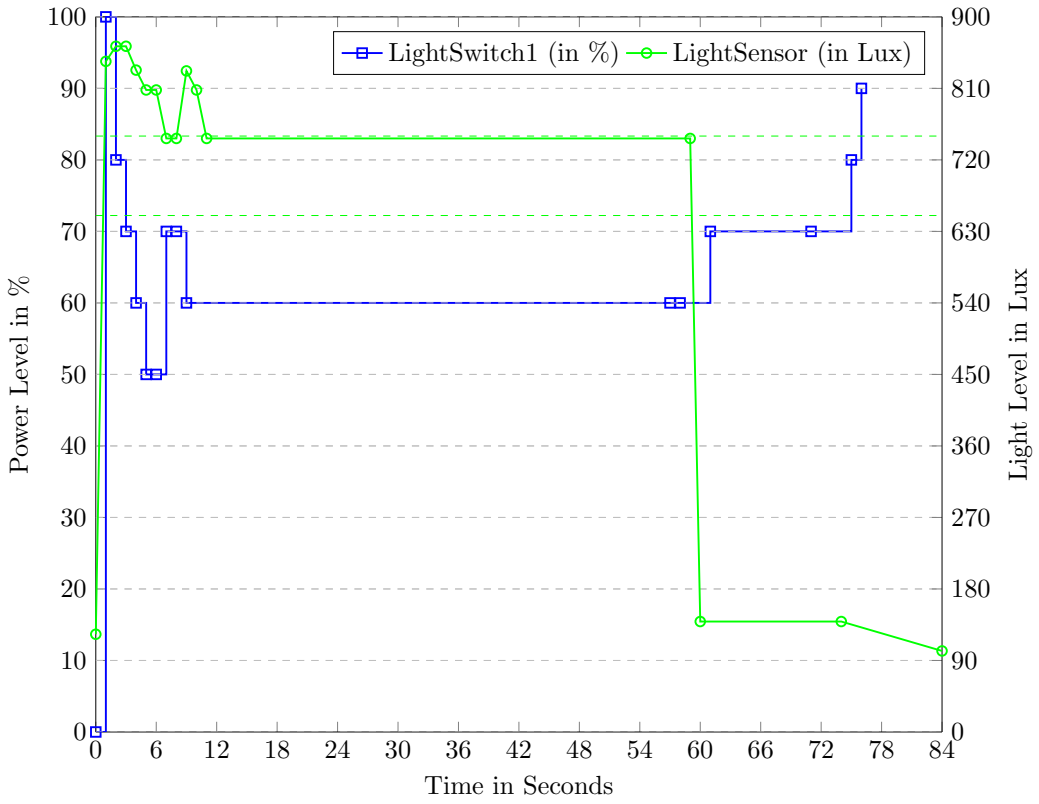


Figure 7.19.: Power Levels of Light Switch and Values from Light Sensor for Baseline Experiment over Time.

off the first lamp manually leading to a sudden drop within the lighting levels. The light control loop detects this change and executes the *IncreaseLight* process step to increase the power level of the same light switch repeatedly (P10–P12) without reaching the optimal lighting conditions defined in the EPL statement. We terminate the whole process instance 76 seconds after its start or it would have continued trying to increase the power levels by repeating the execution of *IncreaseLoop* instances indefinitely.

Table 7.6.: Execution Times for the MAPE-K+ Light Control Process.

ID	Process Step	Duration (in ms)
P1	SwitchOnLight	202
P2	DecreaseLoop	28,085
P3	DecreaseLoop	5,098
P4	IncreaseLoop	73,461
	LightControl	106,846

MAPE-K+ Table 7.6 shows the durations of the executions of the individual process steps of the Light Control process for the MAPE-K+ configuration. Figure 7.20 presents the flow of process executions over time accordingly. The illumination levels

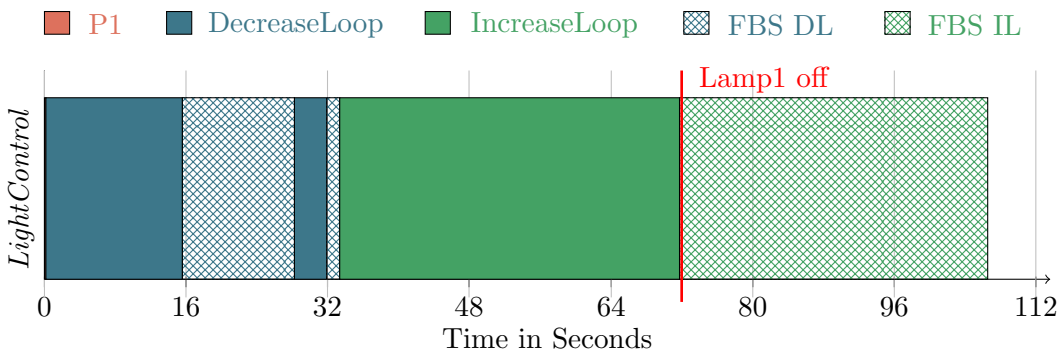


Figure 7.20.: Execution of the Process Step Instances of the MAPE-K+ Light Control Process over Time.

measured by the light sensor as well as the power levels of both light switches are displayed in Figure 7.21. The initial switch on request to the IoT middleware (P1) is executed within similar orders of magnitude as the service requests in previous experiments. This initial request leads to power levels of 100 % and exceeds the light levels defined in the EPL statement, which triggers the light loop to execute the decrease command (P2). For this phase of this particular run of the experiments, the light sensor updated its current values rather slowly, which is why it took some additional time to trigger the first *DecreaseLoop*. Contrary to the *Baseline* process configuration leading to multiple *DecreaseLoop* executions to reach a balance within the light levels, the Feedback Service (*FBS DL*) follows the execution of the first decrease command to fulfil the defined goal for the "DecreaseLight" process step. This goal states that the light levels should fall below 750 Lux within 5 seconds. As the execution of the initial decrease light process step does not lead to these light levels (mismatch: *too high*), the Feedback Service has to initiate multiple planning and execution phases to search for compensation actions to further decrease the light levels. The corresponding *Compensation Query* to derive compensation strategies can be found in Listing 6.2. The result of the Planner based on this query is to execute the "Decrease" (DOWN) command for the light switch once more. After two iterations of the MAPE-K loop decreasing the light levels by dimming down *LightSwitch1* to 70 % the goal was fulfilled successfully. However, the light levels just reached the minimal threshold of 750 Lux to fulfil the specified goal approx. 28 seconds after the process execution started. In the following seconds the threshold of 750 Lux was surpassed again, which triggered the "LightChange" event and another execution of the "DecreaseLight" process step. With that, the Feedback Service was executed to check the light levels again. As the execution of the "Decrease" command for *LightSwitch1* to 60 % was sufficient to reach the desired light levels and fulfil the goal, there was no need to initiate the planning and execute phases.

Approximately 72 seconds after starting the process instance, we manually deactivate the lamp of *LightSwitch1* to simulate a burnt light bulb, which leads to a sudden drop of the light levels. This again triggers the "LightChange" event and, due to the insufficient illumination, to the execution of the "Increase" (UP) command for *LightSwitch1* to a power level of 70 %. With that, the Feedback Service (*FBS IL*)

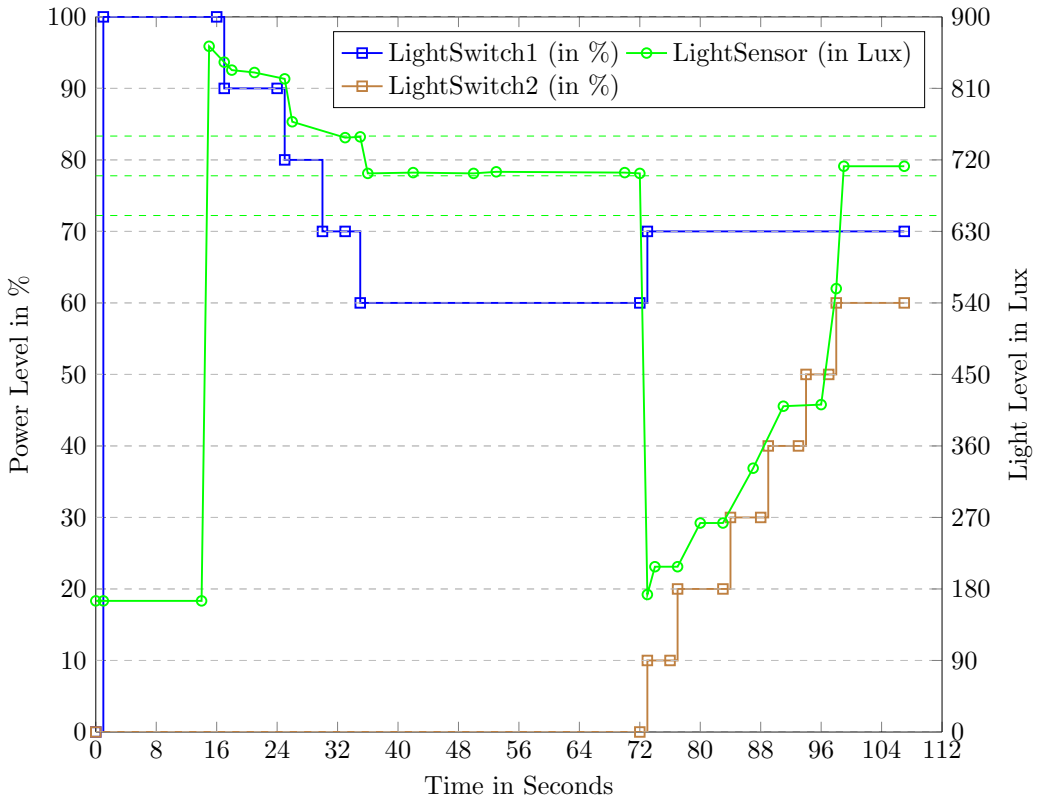


Figure 7.21.: Power Levels of Light Switch and Values from Light Sensor for MAPE-K+ Experiment over Time.

is triggered in order to check and try to reach the goal defined in Listing 7.6 for the *IncreaseLight* process step (i. e., to reach light levels of at least 700 Lux). As the execution of the "Increase" command for *LightSwitch1* did not affect the current values of the light sensor sufficiently within the timeframe of 5 seconds and the compensation condition for the goal is evaluated positively (*violation of Cyber-physical Consistency*), the Feedback Service initiates the planning phase, uses the previously mentioned *Compensation Query* to find a compensation strategy, and executes the compensation actions. For this case, the Planner component derived the execution of the "Increase" (UP) command for *LightSwitch2*–being in the same context and having the same functionality as *LightSwitch1*–as suitable compensation strategy based on the determined mismatch (*too low*). The iterations and execution times of the individual MAPE-K phases and the overall Feedback Service for this particular *IncreaseLoop* instance (P4) can be found in Table 7.7. In total, the Feedback Service analysed 64 relevant data values from the light sensor (*Symptoms*), 6 of which led to the initiation of the Plan phase due to the compensation condition being evaluated positively (i. e., the light level threshold was not reached after 5 seconds). The derived compensation strategy–send "Increase" (UP) command to *LightSwitch2*–was executed 6 times as a result of the Planning phase. The overall execution time for the Feedback Service was 30 seconds for this *IncreaseLoop* to reach lighting levels over 700 Lux by powering up *LightSwitch2* to 60 % (cf. Figure 7.21). After the fulfilment of the goal and with that, the restoration of *Cyber-physical Consistency*,

Table 7.7.: Number of Iterations and Durations of the MAPE Phases and Feedback Service for Process Step P4 of the MAPE-K+ Experiments.

Metrics \ Phase	M	A	P	E	Loop	FB Service
Iterations (#)	–	64	6	6	64	1
Duration (ø in ms)	–	4	1,699	8	1,332	29,925

the process instance was terminated. Due to the asynchronous internal processes of the Feedback Service implementation and the different durations of loops with and without Plan and Execute phases the simple summation of all loop iterations does not yield the overall execution times.

Table 7.8.: Execution Times for the MAPE-K++ Light Control Process.

ID	Process Step	Duration (in ms)
P1	SwitchOnLight	193
P2	DecreaseLoop	3,192
P3	DecreaseLoop	14,219
P4	IncreaseLoop	82,054
P5	IncreaseLoop	55,348
P6	LightError	14,801
P7	DecreaseLoop	15,017
	LightControl	184,824

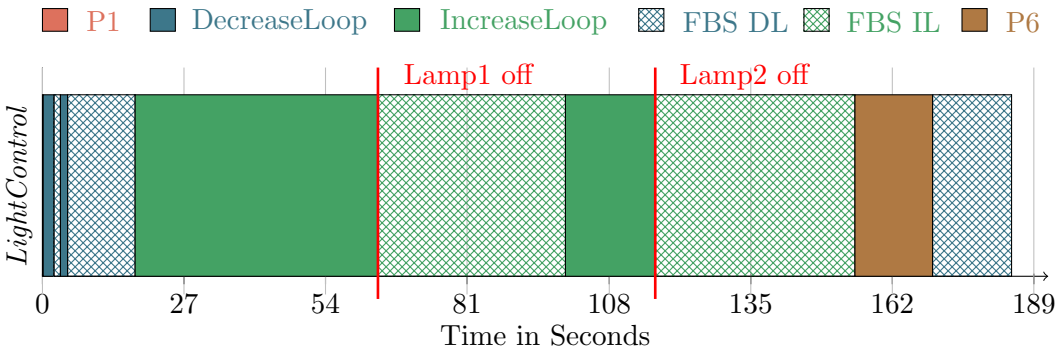


Figure 7.22.: Execution of the Process Step Instances of the MAPE-K++ Light Control Process over Time.

MAPE-K++ Table 7.8 shows the durations of the executions of the individual process steps of the Light Control process for the MAPE-K++ configuration. Figure 7.22 presents the flow of process executions over time accordingly. Similar to the MAPE-K+ configuration, the initial activation of the *LightSwitch1* (P1) is followed by multiple *DecreaseLoops* (P2–P3) including the execution of the Feedback Service (*FBS DL*) to reach the desired lighting levels. Figure 7.23 shows the corresponding

values of the light sensor and the power levels of both light switches used in this experiment. A balance within the light levels was reached about 18 seconds after the start of the process instance at a power level of 60 % for *LightSwitch1*. We waited approx. 46 seconds to deactivate the lamp of *LightSwitch1* manually, which caused the execution of the "IncreaseLight" process step for *LightSwitch1* as part of the *IncreaseLoop* (P4). As the "Increase" (UP) command does not lead to a change within the light levels, the Feedback Service (*FBS IL*) is executed additionally for this *IncreaseLoop* to restore *Cyber-physical Consistency* by means of dimming up *LightSwitch2* to 60 % similar to the MAPE-K+ experiment. The execution of the MAPE-K loops by the Feedback Service for this process step took approx. 36 seconds to reach the specified goal of at least 700 Lux.

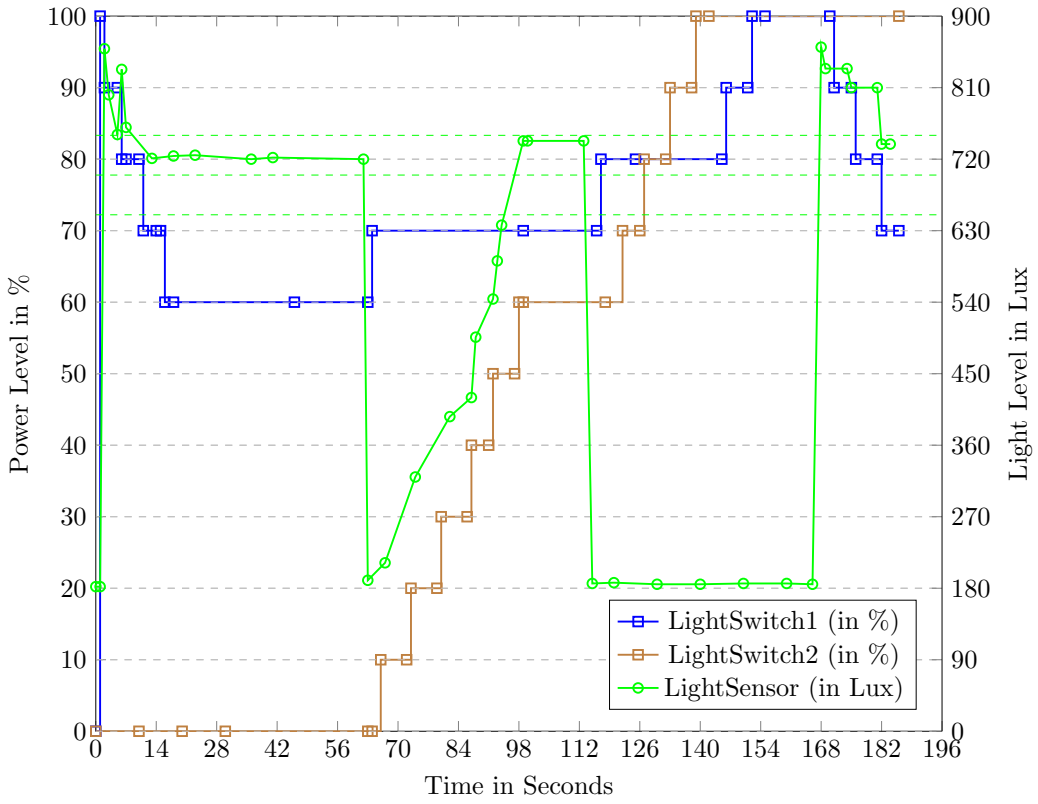


Figure 7.23.: Power Levels of Light Switch and Values from Light Sensor for MAPE-K++ Experiment over Time.

Table 7.9.: Number of Iterations and Durations of the MAPE Phases and Feedback Service for Process Step P5 of the MAPE-K++ Experiments.

Metrics \ Phase	M	A	P	E	Loop	FB Service
Iterations (#)	–	67	6	6	67	1
Duration (ø in ms)	–	3	13	2	844	38,092

For the MAPE-K++ run, we also deactivate the lamp of *LightSwitch2* to simulate another error with the lighting system–approx. 17 seconds after *LightSwitch2* restored cyber-physical consistency. This triggered another *IncreaseLoop* instance (P5) including the execution of the Feedback Service. The relevant statistics for this Feedback Service execution can be found in Table 7.9. First, the Feedback Service tried to increase the power levels of *LightSwitch2* from 60 % to 100 % in four Planning and Execution phases (cf. *Compensation Query* in Listing 6.2 and Goal in Listing 7.6). As this did not lead to the desired light levels and the *too low* mismatch was still present, the Feedback Service powered up *LightSwitch1* from 80 % to 100 % in two iterations as well. The goal could still not be fulfilled by these actions and the Feedback Service was not able to derive any more compensation strategies, which is why it reported back the goal as *unsatisfied* to the *PROtEUS* system. All in all, the Feedback Service took approx. 38 seconds to execute 67 MAPE-K loops with 6 Plan and Execute phases for this *IncreaseLoop* (P5). The result of the Feedback Service reported back to PROtEUS led to the activation of a *Failure Port* that is part of the definition of the "IncreaseLight" process step (cf. Figure 7.16). The execution of the corresponding failure branch initiated the human task "LightError" (P6) to be sent to the resident's mobile interaction device to inform him/her about the broken lights. The process engine then waits for a response from the user. Approximately 15 seconds after this notification, we manually reactivate the lamp of *LightSwitch1* and confirm the human task, which causes the process execution to continue. As the power level of *LightSwitch1* is still at 100 % the light levels are now too high again, which triggers another *DecreaseLoop* (P7) to dim down *LightSwitch1* to 70 % with the help of the Feedback Service to reach the optimal lighting conditions again after about 15 seconds. We then terminate the process. The power levels of *LightSwitch2* are still at 100 % as its lamp is still deactivated.

Discussion

With these comprehensive experiments regarding the Smart Lighting scenario, we show the interaction of the *PROtEUS* WfMS with the *Feedback Service* and its benefits regarding the detection and autonomous repair of cyber-physical errors (Requirements *R5* and *R6*) based on the self-management capabilities of the Feedback Service (Requirement *R7*). The *Baseline* experiments are used to show the lack of functionality of the IoT devices (dimmer switches) and the basic workflow system to detect the cyber-physical errors and violations of cyber-physical consistency regarding the actual and assumed lighting levels. This leads to an infinite loop of powering up *LightSwitch1* and executing the *IncreaseLoop* indefinitely, which is rather undesired behaviour.

The *MAPE-K+* experiment shows the benefits of the Feedback Service to verify the process execution and also to repair errors autonomously. We use the light sensor as an additional information source that can link the process execution to its physical effects. The Feedback Service evaluates the data from this sensor and uses the context model as well as the compensation query to derive suitable compensation strategies to be executed to restore cyber-physical consistency. This self-healing mechanism is currently based on adapting or exchanging the process resources. From the *MAPE-K+* results, we see that the Feedback Service takes more control over the executions of the process compared to the *Baseline* experiments. The Feedback

Service's attempts to reach the defined goals by adapting the power levels of the light switches replace the additional iterations of the *IncreaseLoop* and *DecreaseLoop* executions within the Baseline workflow. The computational overheads introduced by the Feedback Service's MAPE-K loops are in the same order of magnitude as the operations executed by the PROtEUS base system. The activities in the physical world are again significantly slower than the computations within the Feedback Service. Although we did not specify any *Consistency Levels* for this experiment, their application can reduce the number of required MAPE-K iterations to reach the lighting thresholds defined in the goals. This can contribute to a reduction of the overall execution time for the Feedback Service and the underlying CPS workflow.

The *MAPE-K++* is experiment is used to illustrate the error handling mechanisms of the PROtEUS base system in combination with the Feedback Service (Requirements *R5* and *R6*) as well as the integration of human interactions in the workflows (Requirement *R3*). At the point of the Feedback Service not being able to find suitable compensations anymore, it reports its termination and the *unsatisfied* goal back to the WfMS. The process step's failure port and its attached failure branch can be used to model a generic way of handling errors that cannot be resolved by the Feedback Service–in our case to notify the user and wait for his/her actions to remedy the problem manually. In our experiment, the user has to confirm the execution of the human task manually for the process instance to continue. Adding the "cyberPhysical" attribute and a similar goal (cf. Listing 7.6) to the human task defining the required light levels in the *satisfied condition* can trigger the Feedback Service for this process step instance again, and automatically detect when the user performed the compensation actions successfully.

7.4.3. Robot Navigation Process

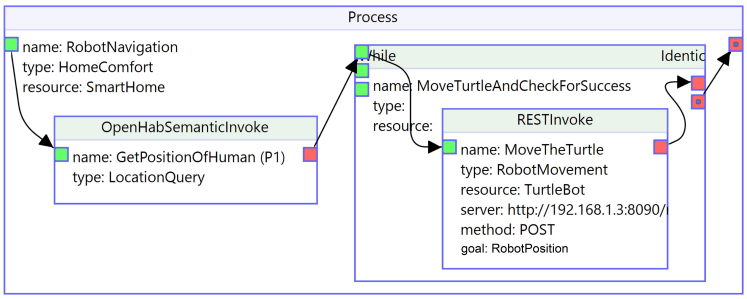


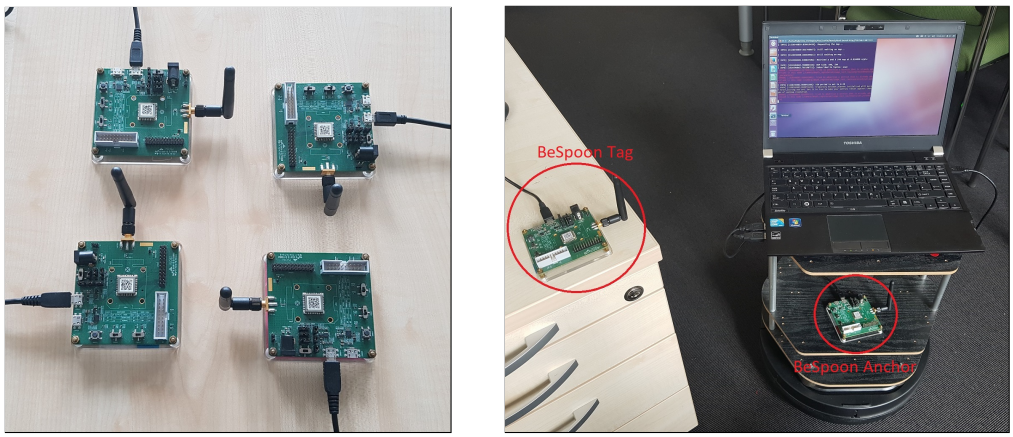
Figure 7.24.: Robot Navigation Process.

In this experiment, we investigate the aspect of using the autonomous driving and navigation functionality of the TurtleBot service robots based on SLAM from within a process (cf. *Morning Routine* scenario in Section 2.2.1) [SHA18b]. As already discussed, the internal navigation processes of the robots are very fragile leading to the robot getting stuck relatively quickly and assuming wrong coordinates for its current location determined by the SLAM algorithms. We will use an external indoor location application based on the *BeSpoon*[16] tracking system as external

[16] `http://bespoon.com/`

data source for the Feedback Service to verify the correct position of the robot. The process goal is to ensure a correct physical positioning of the robot in the target area despite inaccuracies and various sources for errors and vulnerabilities within the robot's internal navigation system. This experiment is similar to the workflow-based use case described in [MMS14], where GPS data is used to automatically verify the position of people in a rescue scenario.

Experiments



(a) BeSpoon Tracking System. (b) TurtleBot 2 Robot with BeSpoon.

Figure 7.25.: Hardware Setup for Robot Navigation Experiments.

The underlying process model "RobotNavigation" can be found in Figure 7.24. The aim is to send the robot to a specific location in the room based on the position of the user. First, the position of the user is retrieved via a semantic select process step from the middleware (*GetPositionOfHuman*, P1). The corresponding SPARQL query can be found in Listing 7.1. The coordinates received from the IoT middleware will then be used as target coordinates for the robot movement process. The following loop *MoveTurtleAndCheckForSuccess* sends the robot to these specific coordinates via a REST service call to the middleware (*MoveTheTurtle* process step). This process step contains an active *cyberPhysical* flag and the goal attribute *Robot Position* specified in Listing 7.7.

The satisfied condition of the objective refers to the coordinates of the external BeSpoon tracking system. We implemented a custom service for ROS (*BeSpoonROS*) that publishes data from the BeSpoon system as a dedicated ROS topic. The BeSpoon system (cf. Figure 7.25(a)) is set up using three nodes placed statically in the room and the anchor mounted on TurtleBot2 and connected to the TurtleBot's control laptop (cf. Figure 7.25(b)). BeSpoon uses trilateration based on the distances between the nodes and the moving anchor to determine the position of the anchor and with that, the position of the TurtleBot. The BeSpoonROS service transforms the BeSpoon coordinates and the robot's internal coordinates into a common simplified coordinate system to make them comparable. This step is necessary due to the robot's internal coordinates being very precise and highly fluctuating, which prevents a check for equality with the BeSpoon coordinates. The basis of the simplified

Listing 7.7: Goal and Objective for MoveTheTurtle Process Step.

```
1 "MoveTheTurtle" : {
2 "name":"Robot Position",
3 "objectives":[
4  { "name":"robot reached the desired position",
5    "satisfiedCondition":"#robotReachedPosition(#position,
         5.8D , 13.0D , 1.0D) == true",
6    "compensationCondition":"#movement.contains('ARRIVED')
       ",
7    "contextPaths":[
8     "MATCH(posNode {name: '
         State_proteus_turtle_simplePosition_1'})",
9     "MATCH(posNode)-[:hasStateValue]-(posValue)",
10    "MATCH(moveNode {name: '
         State_proteus_turtle_movement_1'})",
11    "MATCH(moveNode)-[:hasStateValue]-(moveValue)",
12    "RETURN id(posNode) as stateId, posValue.
         realStateValue as position, moveValue.
         realStateValue as movement"
13 ] } ] }
```

coordinate system is a clustering of the respective room into squares of 25 x 25 cm. The destination of the robot (position of the user) is at coordinates $x = 5.8$ and $y = 13.0$. Within the satisfied condition, we use these coordinates to specify a criterion for the successful process step execution (Line 5). The *robotReachedPosition* function is a custom function that checks if the current position (as defined in the context path; Lines 7–13) is within the vicinity of the specified target coordinates. Due to the precision of the BeSpoon system and also some fluctuation, the fourth argument of the function defines a range of ± 1.0 for each coordinate to be fulfilled. The compensation condition states that if the robot's internal navigation processes publish an "ARRIVED" event to the middleware (Line 6) indicating that the robot has reached its destination based on its internal assumption of its position and the satisfied condition was not evaluated positively, the Feedback Service should terminate with the goal being *unsatisfied*. The PROtEUS system checks the state of the goal within the "MoveTurtleAndCheckForSuccess" process loop condition and initiates another loop iteration if the goal is not satisfied, i. e., it executes another instance of the "MoveTheTurtle" process step with the initial destination as input parameter and the Feedback Service checking if the desired position is reached based on the simplified BeSpoon coordinates. The goal will only be fulfilled if the satisfied condition becomes true (i e., the external BeSpoon coordinates confirm the correct position). We extended the *DROiT API* [SSAS15] to also make the simplified positions of the robots based on the internal SLAM system and the external BeSpoon system accessible via the middleware. These simplified coordinates were also added as additional location attributes to the semantic models of humans and robots in the knowledge base (cf. Section 4.3.3).

After starting the first loop iteration and initiating the "MoveTheTurtle" driving process including the parallel invocation of the Feedback Service for this process

Figure 7.26.: Path of TurtleBot for RobotNavigation Process (Black Lines = Obstacles/Walls, White Area = Discovered, Grey Area = Unknown).

step, we wait a few seconds to put a box as an obstacle in the way of the robot. As it is moving within a narrow space, the robot is not able to circumnavigate the obstacle, which results in the internal navigation processes to terminate and the robot to publish an "ARRIVED" event. This leads to the Feedback Service reporting the *unsatisfied* goal back to the workflow engine as the final destination defined in the satisfied condition was not reached (as confirmed by the external BeSpoon system). The process then restarts the "MoveTheTurtle" process step within the "MoveTurtleAndCheckForSuccess" loop. We remove the obstacle and the robot is instructed to repeat the driving process. After some more iterations and "ARRIVED" events published by the robot due to errors and obstacles, it arrives at its destination as defined and confirmed by the simplified BeSpoon coordinates eventually. Figure 7.26 shows the robot's map and its path from the random starting to its destination after the repeated execution of the "MoveTheTurtle" process step to remedy errors that occurred during driving. The distance from the random starting point to the human is approx. 4.5 meters.

Results

Table 7.10 shows the durations of the executions of the individual process step instances of the robot navigation process. Figure 7.27 presents the flow of process executions over time accordingly. First, the target position for the robot to drive to is fetched from the IoT middleware using a semantic invoke process step (P1). The PROtEUS system then uses this parameter to initiate the driving process within the "MoveTurtleAndCheckForSuccess" loop (P2). Approximately 5 seconds after the start of the TurtleBots's driving processes we put a large obstacle in its way, which led to a replanning of the path to the destination. However, due to the obstacle and narrow space, the robot cancelled the replanning after approx. 23 sec-

Table 7.10.: Execution Times for Process Steps of the RobotNavigation Process.

ID	Process Step	Duration (in ms)
P1	GetPositionOfHuman	1,079
P2	MoveTheTurtle (1)	22,924
P3	MoveTheTurtle (2)	7,530
P4	MoveTheTurtle (3)	10,065
P5	MoveTheTurtle (4)	7,759
	MoveTurtleAndCheckForSuccess	48,278
	RobotNavigation	49,357

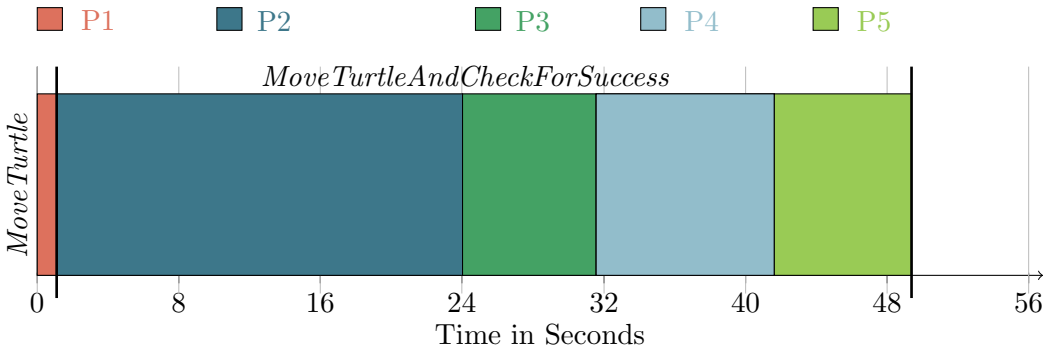


Figure 7.27.: Execution of the Process Step Instances of the RobotNavigation Process over Time.

onds. This triggered an "ARRIVED" event to be published by the middleware. The Feedback Service running the MAPE-K loops for the "MoveTheTurtle" process step detected this event, evaluated the compensation condition positively as the current BeSpoon position did not correspond to the criterion defined in the satisfied condition, and initiated the planning phase. The number of MAPE-K iterations, individual phases and durations of the Feedback Service execution for each repetition of the *MoveTheTurtle* process step can be found in Table 7.11. The current coordinates of the TurtleBot according to the BeSpoon tracking system at the time of publishing an *ARRIVED* event based on the TurtleBots internal localization are shown in Figure 7.28.

Table 7.11.: Number of Iterations and Durations of the Individual MAPE Phases and Feedback Service for the RobotNavigation process steps.

ID	Process Step	#M	#A	#P	#E	FBS Duration (in ms)
P2	MoveTheTurtle (1)	–	17	1	1	21,084
P3	MoveTheTurtle (2)	–	6	1	1	7,202
P4	MoveTheTurtle (3)	–	9	1	1	10,028
P5	MoveTheTurtle (4)	–	6	0	0	7,333

For each repetition (P2–P5) of the *MoveTheTurtle* process step, the Feedback Service is executed in parallel once. Within the first loop execution, the Feedback Ser-

vice analysed 17 sensor values (i. e., the movement status and position data according to the context paths in Listing 7.7). After the detection of the *ARRIVED* event by the Feedback Service's Analyser component, the Planner was not able to find any compensation, which led to the Feedback Service's Executor communicating its termination and the unsatisfied goal back to the PROtEUS WfMS. The status of the goal is mapped to an outgoing data port of the "MoveTurtleAndCheckForSuccess" process loop and used as stop criterion for this loop. If the goal is left *unsatisfied*, the loop will be repeated and the robot is instructed to drive to the same target location again. In the following loop iterations (P3–P5), the robot again had some troubles with its internal positioning, getting stuck at different obstacles or loosing its current internal location. After three more iterations and lengthy reorientations taking approx. 25 seconds in total, the robot arrived at its destination approx. 48 seconds after beginning to move to the target location. At 48 seconds and the ARRIVAL of the robot, the external BeSpoon coordinates were within the ranges defined by the goal to fulfil the objective with respect to the satisfied condition (cf. Figure 7.28). The corresponding MAPE-K instance of the Feedback Service published the *satisfied* goal back to PROtEUS and the *RobotNavigation* process terminated as the stop criterion for the *MoveTurtleAndCheckForSuccess* loop was evaluated positively. The robot's final path for this experiment can be found in Figure 7.26.

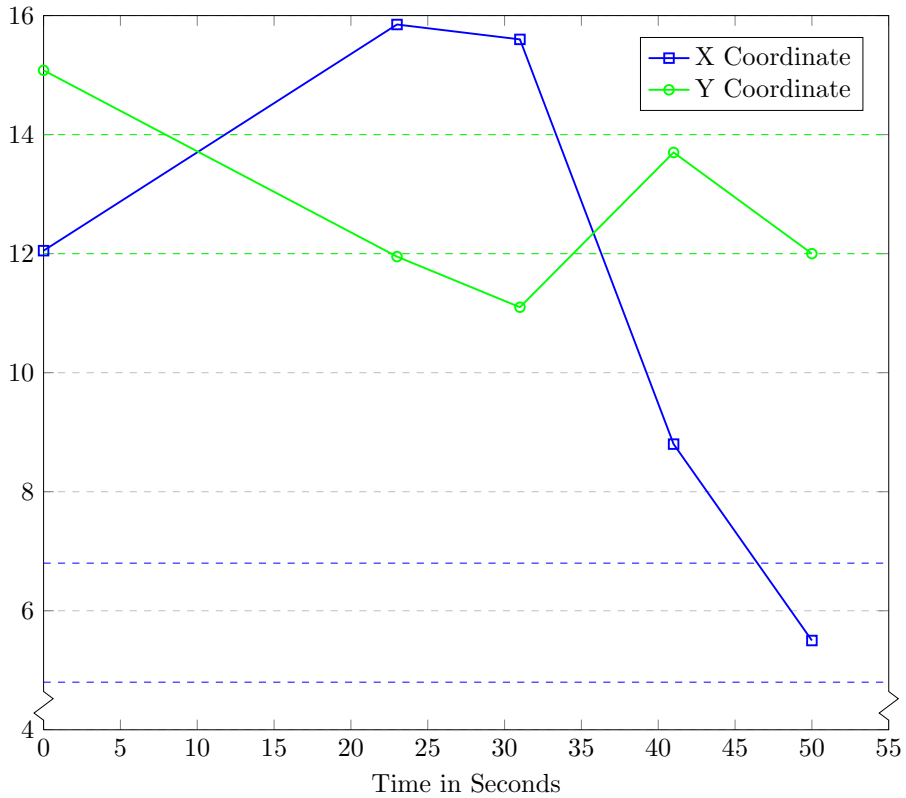


Figure 7.28.: Coordinates of the TurtleBot Robot Movement for the RobotNavigation Process over Time.

Discussion

This experiment shows the interactions among the PROtEUS base system, Feedback Service, openHAB middleware and TurtleBot robots. As the robot's internal navigation processes are very fragile and imprecise, we use an external indoor tracking system as additional data source to verify the robot's correct location (Requirement *R5*) and detect possible *cyber-physical inconsistencies* (i. e., deviations within the robot's assumed location and its actual location). We designed the Robot Navigation process in a way that the result of the Feedback Service executions is used to decide whether to reinstruct the robot to repeat the execution of a certain process step or to terminate the successful process. For the experiment, this strategy seems suitable as there is no need to actually replace the "broken" robot and send another robot instead, which would have been the compensation strategy of the Feedback Service using the compensation query for cyber-physical consistency (cf. Listing 6.2) with available replacement actuators. However, as there was no instance of a second robot to be found in the knowledge base, the Feedback Service terminates not being able to satisfy the goal, which leads to the repetition of the driving process. This reinstantiation helped the robot in all cases to find its current position again after taking some time to reorientate itself using its internal SLAM algorithms. The example shows that the CPS processes have to be designed according to the use case, desired behaviour, context and available resources.

It is worth noting that the external BeSpoon tracking system is also subject to imprecisions and fluctuations as it is highly influenced by other objects and electromagnetic fields. It does not help to increase the precision of the robot's internal system. However, we can use these tracking sensors as external data sources to detect major deviations from the TurtleBot's assumption of its current location based on the internal SLAM processes. For the use case of driving to the paper boy to have the paper put on the robot (cf. *Morning Routine process* in Section 2.2.1), a lower degree of precision should be sufficient to successfully execute the process. The extended use case ("Retrieve Insulin Injection") for distributed process execution described in Section 5.5 requires the robot to drive to a certain location to pick up an insulin injection from a shelf. Here, a higher degree of precision regarding the robot's location and with that, a better indoor tracking system are necessary.

7.4.4. Distributed Robot Driving Process

In the following experiments, we use the *PROtEUS* base system and the *D-PROtEUS* system in combination with the Feedback Service in a distributed process execution setup (Requirement *R4*). The Feedback Service is used to manage the process execution on the *Peers* (here: robots) with respect to different criteria (Requirement *R7*): the *Liveliness* of a peer as well as its *Battery Levels*. The simple scenario process "MoveTurtle" is depicted in Figure 7.29. It contains a subprocess "MoveTurtleRemote" that has an active *distributed* flag and *resource* attribute. This subprocess is supposed to be transferred to the given resource (*TurtleBot1* as identified by its IP address) to be executed on this peer. The *MoveTurtleRemote* subprocess contains one process step "Drive" to invoke a specific ROS [QCG+09] service on the robot locally, which instructs the robot to simply move forward one step. The experiments are based on the elaborations in [SNS14b, SHA17] and Section 6.4.

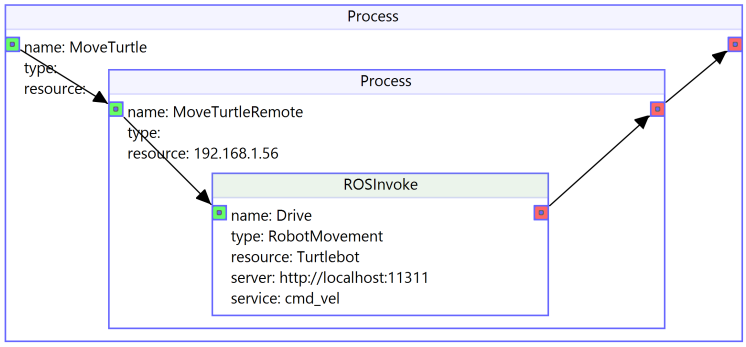


Figure 7.29.: Distributed MoveTurtle Process.

Figure 7.30.: TurtleBot Robots used in Distributed Execution Experiments.

Experiments

We run an instance of the *MoveTurtle* process on the PROtEUS system of the control computer set up as a *Super-Peer* with an active *Distribution Manager* component (*D-PROtEUS*). Two TurtleBot robots (cf. Figure 7.30) running the basic configuration of the PROtEUS WfMS act as regular *Peers* that are registered with and managed by the super-peer. The Feedback Service (FBS) is running in parallel to the D-PROtEUS system on the super-peer (cf. Figure 7.31). As described in Section 5.5, we use *Subcontracting* [vdA00] and *Instance Migration* [ZHKL10] as mechanisms to transfer process fragments to peers and their responses/results back to the super-peer. Upon its execution by the super-peer, the "MoveTurtleRemote" subprocess is sent to *TurtleBot1* identified by its IP address as defined by the *Resource* attribute of the subprocess. The Feedback Service then is initiated in parallel to check for *Peer Liveliness* and *Battery Levels* by the super-peer.

Peer Liveliness In this experiment the super-peer periodically checks the liveliness of the peer. The robots are good examples for mobile CPS devices with an unreliable network connection. They move within the smart space while being connected to the wireless local area network. When leaving the area of WiFi coverage, the robot

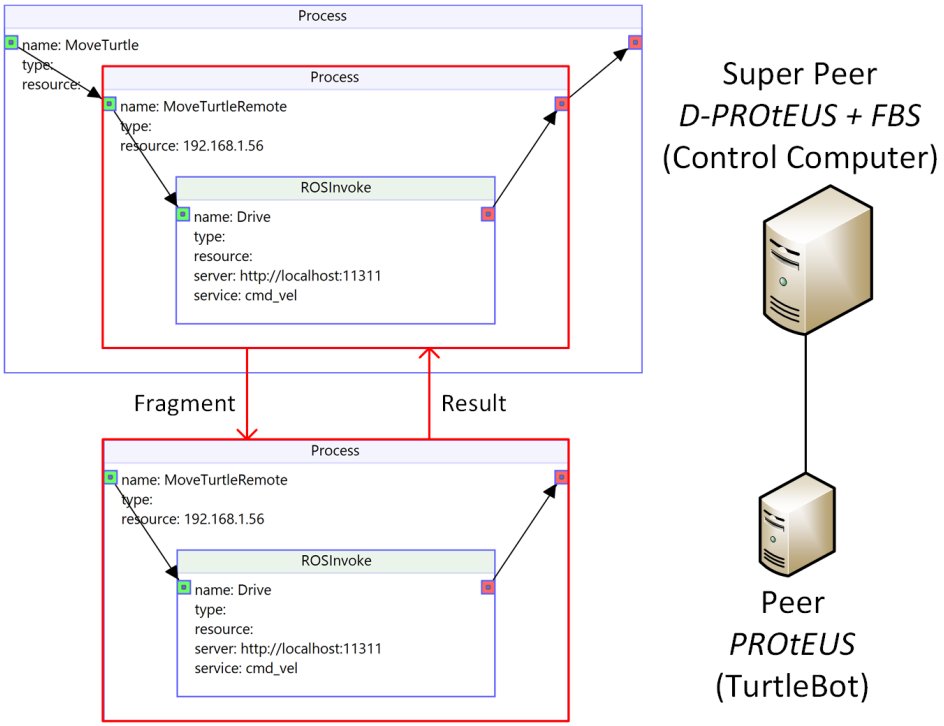


Figure 7.31.: Distributed Process Execution Setup.

becomes disconnected and is likely to not be able to reconnect or to fail executing the assigned processes. For that reason, we check the robot's (peer's) liveliness signal periodically while executing the *MoveTurtleRemote* subprocess instance. If the execution takes longer than 15 seconds and the last signal regarding the process execution has been received 15 seconds ago or longer, the super-peer assumes an error regarding this peer and tries to find an alternative peer to repeat the distributed execution of the subprocess on. This scenario is similar to the use cases described in [PRS+13] referring to the offline distribution of workflow tasks with unreliable connections; and in [DTB+15] referring to the periodic check of liveliness of workflow resources and the search for substitutes after a timeout.

Listing 7.8 shows the goal attribute "process executed on peer" of the *MoveTurtleRemote* subprocess regarding the *Peer Liveliness* setup. The context paths are used to retrieve the state of the subprocess based on its identifier (ID) and the ID of the executing peer (Lines 7–11). The satisfied condition defines the success criterion as the subprocess being in state "executed" (Line 5). On the other hand, the compensation condition refers to the objective being created more than 15 seconds ago and the last command being sent to the peer related to this objective more than 15 seconds ago (Line 6). With this compensation condition, we assume that the remote execution of the *MoveTurtleRemote* subprocess should not take longer than 15 seconds and that the super-peer should be able to successfully poll the peer (TurtleBot1) for status updates within this time frame. The custom *lastCommandSendBefore* function implements this second factor. In this setup, the super-peer polls the status from the TurtleBot1 peer every second. The compensation query used to find an alternative peer to repeat the subprocess execution on is described

Listing 7.8: Goal and Objective for the MoveTurtleRemote Process Step Regarding Peer Liveliness.

```
"MoveTurtleRemote" : {
"name":"process executed on peer",
"objectives":[
{ "name":"process executed within 15 seconds",
  "satisfiedCondition":"#state == 'executed'",
  "compensationCondition":"#objective.created.isBefore(#
     now.minusSeconds(15)) && #lastCommandSendBefore(#
     objective, 15)",
  "contextPaths":[
   "OPTIONAL MATCH (n:NeoProcess {processId:{piid}})",
   "RETURN",
   "CASE WHEN n IS NOT NULL THEN n.state ELSE 'unknown'
      END AS state,",
   "CASE WHEN n IS NOT NULL THEN id(n) ELSE 0 END AS
      nodeId"
] } ] }
```

Listing 7.9: Goal and Objective for the MoveTurtleRemote Process Step Regarding Peer Battery Levels.

```
"MoveTurtleRemote" : {
"name":"move turtlebot with enough battery power",
"objectives":[
{ "name":"enough power for current executing turtlebot",
  "satisfiedCondition":"#state == 'executed'",
  "compensationCondition":"#batteryValue < 50 && #
     lastCommandSendBefore(#objective, 10)",
  "contextPaths":[
   "OPTIONAL MATCH (process:NeoProcess {processId: {piid
      }}) ",
   "OPTIONAL MATCH (peer1:NeoPeer {name: 'turtlebot2-
      X201EP'})-[:HAS_METRIC]->(metric1:NeoPeerMetric)",
   "OPTIONAL MATCH (peer2:NeoPeer {name: 'tb-Satellite-
      R630'})-[:HAS_METRIC]->(metric2:NeoPeerMetric)",
   "OPTIONAL MATCH (process)<-[:REMOTE_FOR]-(remote:
      NeoProcess)-[runsOn2:RUNS_ON]->(peer2)",
   "WITH process, remote, ",
   "CASE WHEN runsOn2 IS NOT NULL THEN peer2 ELSE peer1
      END AS peer, ",
   "CASE WHEN process IS NOT NULL THEN id(process) ELSE 0
       END AS nodeId, ",
   "CASE WHEN runsOn2 IS NOT NULL THEN metric2 ELSE
      metric1 END AS metric ",
   "RETURN remote.state AS state, peer, nodeId, ",
   "CASE WHEN metric IS NOT NULL AND metric.hasBattery
      THEN metric.batteryValue ELSE 100 END AS
      batteryValue"
] } ] }
```

in detail in Section 6.4. We prepared the first TurtleBot (Peer1) to not be able to respond to the status update requests from the super-peer (SP) by shutting down the peer's PROtEUS WfMS immediately after receiving the *MoveTurtleRemote* subprocess. This will lead to the Feedback Service running on the super-peer to detect an issues with Peer1 and to find *TurtleBot2* (Peer2) as replacement peer to redeploy the subprocess on after the timeout defined in the compensation condition.

Peer Battery Levels In this experiment we use a process and peer–super-peer setup similar to the *Peer Liveliness* configuration. The super-peer running the D-PROtEUS system and Feedback Service instructs the peer *TurtleBot1* to execute the *MoveTurtleRemote* subprocess including the *Drive* process step invoking the driving functionality of the robot. In parallel, the Feedback Service running on the super-peer checks the battery level of the robot peer. In case the first peer's (Peer1) battery levels fall below 50 %, the subprocess is redeployed and executed on the second available TurtleBot peer (Peer2). This scenario corresponds to the work described in [CLD+15] where entities collaborate automatically if the current device's battery levels fall below a certain threshold.

Listing 7.9 shows the goal attribute for the *MoveTurtleRemote* subprocess regarding the *Peer Battery Levels* setup. The satisfied condition defines the subprocess being in state "executed" as criterion for the successful process execution (Line 5). The compensation condition states that the Plan phase should be initiated if the battery levels of the current peer executing the subprocess fall below 50 % and the last status update regarding the process execution has been received 10 or more seconds after starting the MAPE-K loops by the Feedback Service for the instance of the *MoveTurtleRemote* subprocess (Line 6). This indicates a potential issue with the robot executing the subprocess as the battery is draining but the process execution is not progressing. The context paths (Lines 7–17) are used to retrieve the current robot's battery levels–identified by the corresponding host name of the peer associated with the subprocess instance that is running on this peer. This context path definition is a good example for using more advanced features of the *Cypher* query language (e. g., *OPTIONAL MATCH* and *CASE WHEN* statements) [Web12]. In case the compensation condition can be evaluated positively by the Feedback Service, the compensation query described in Section 6.4 is used to find an alternative peer (*TurtleBot2*) and redeploy the subprocess on this peer. We set up the first robot to start with a battery level of over 50 % and drop under 50 % during execution of the *Drive* process step, which is configured to take approx. 10 seconds. During the execution of this process step, the super-peer will not receive any status updates regarding the process execution. This will eventually lead to the Feedback Service assuming an error and searching for an alternative peer and reinstantiating the *MoveTurtleRemote* subprocess on this peer. The robots' battery levels are checked periodically by the Feedback Service. They are published to the knowledge base and middleware as sensor data via the *DROiT API*.

Results

Peer Liveliness The execution times for the process step instances of the *MoveTurtle* process regarding the Peer Liveliness experiment are shown in Table 7.12. The corresponding flow of executions over time can be found in Figure 7.32. The

Table 7.12.: Execution Times for Process Steps of the MoveTurtle Process Regarding the Peer Liveliness.

ID	Process Step	Duration (in ms)
P1	MoveTurtleRemote (Peer1)	15,862
P2	MoveTurtleRemote (Peer2)	2,427
P3	Drive (Peer2)	2,409
	MoveTurtle (SP)	18,289

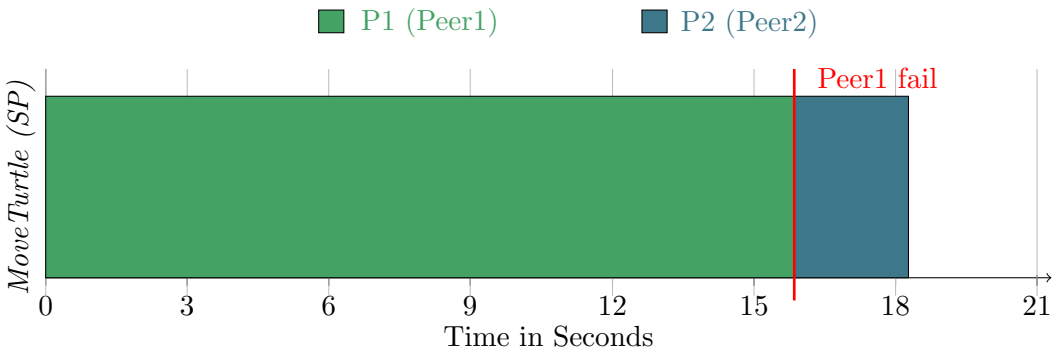


Figure 7.32.: Execution of the Process Step Instances of the MoveTurtle Process Regarding the Peer Liveliness Experiment over Time.

process starts with the super-peer (SP) WfMS instantiating the *MoveTurtle* process model. Due to the "Distributed" flag and resource attribute pointing to the IP address of *TurtleBot1* (Peer1), the *Distribution Manger* called the *Process Manager* of the PROtEUS WfMS running on Peer1 to execute the subprocess (P1). The Feedback Service was initialized with the goal attribute for the *MoveTurtleRemote* process step on the super-peer in parallel to the distribution of the subprocess. The statistics of the MAPE-K executions for process step P1 are shown in Table 7.13. After analysing 15 sensor values regarding the status of the subprocess execution on Peer1 and approx. 15 seconds of not receiving any new status updates from the peer, the criterion defined in the compensation condition became true (*Peer1 fail*). As a result, the Planner determined a new peer to reinstantiate the subprocess on (cf. Section 6.4). The Executor instructed the super-peer WfMS to repeat the execution of the subprocess with the same parameters on *TurtleBot2* (Peer2). The Distribution Manager of the super-peer invoked the Process Manager of Peer2 to execute the subprocess (P2) and with that, the execution of the MAPE-K loops by the super-peer for this process step, again. Peer2 executed the subprocess successfully within approx. 2 seconds, which led to the satisfied assumption to become true and the Feedback Service terminating with the successful execution of P2 on Peer2 and of the *MoveTurtle* process on the super-peer (SP) after approx. 18 seconds. As the *Drive* process step was not executed successfully on Peer1, we do not list it in Table 7.12.

Table 7.13.: Number of Iterations and Durations of the MAPE Phases and Feedback Service for Step P1 of the Peer Liveliness Experiments.

Metrics \ Phase	M	A	P	E	Loop	FB Service
Iterations (#)	–	15	1	1	15	1
Duration (ø in ms)	–	5	17	44	1,018	15,455

Table 7.14.: Execution Times for Process Steps of the MoveTurtle Process Regarding the Peer Battery Levels.

ID	Process Step	Duration (in ms)
P1	MoveTurtleRemote (Peer1)	10,325
P2	MoveTurtleRemote (Peer2)	2,420
P3	Drive (Peer2)	2,408
	MoveTurtle (SP)	12,745

Peer Battery Levels The execution times for the process step instances of the *MoveTurtle* process regarding the Peer Battery Level experiment are shown in Table 7.14. The corresponding flow of executions over time can be found in Figure 7.33. The process execution starts in a similar way as the previous experiment: the super-peer distributes the *MoveTurtleRemote* process to Peer1, where it is executed (P1) while being monitored by the Feedback Service from the super-peer. As defined in the respective goal for the distributed subprocess, the Feedback Service triggered the Planner to search for a compensation approx. 10 seconds after not receiving any status updates regarding the process execution and the battery levels for Peer1 dropping below 50 % (cf. Figure 7.34). The statistics of the MAPE-K executions for process step P1 are shown in Table 7.15. The Feedback Service analysed 9 sensor values regarding the state of the battery levels of the current peer, but it did not receive any execution status updates. After 10 seconds and the battery levels falling below the defined threshold (*Peer1 fail*) the Feedback Service finds the alternative peer and reinstantiates the subprocess in a way similar to the Peer Liveliness experiment. On Peer2, the execution of the *MoveTurtleRemot* process (P2) including the *Drive* process step (P3) takes approx. 2 seconds, which is communicated to the Feedback Service as an execution status update (*executed*). The overall *MoveTurtle* process was executed on the super-peer (SP) after approx. 13 seconds. As the *Drive* process step was not successful on Peer1, we do not list it in Table 7.14.

Table 7.15.: Number of Iterations and Durations of the MAPE Phases and Feedback Service for Step P1 of the Peer Battery Levels Experiments.

Metrics \ Phase	M	A	P	E	Loop	FB Service
Iterations (#)	–	9	1	1	9	1
Duration (ø in ms)	–	9	28	38	962	9,975

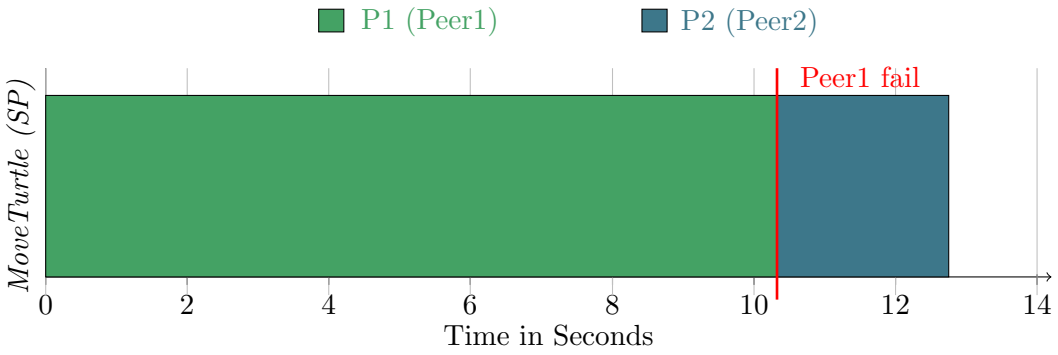


Figure 7.33.: Execution of the Process Step Instances of the MoveTurtle Process Regarding the Peer Battery Levels Experiment over Time.

Error Cases for Distributed Process Execution

Experiments We conducted further experiments with respect to the distributed execution of processes using the two TurtleBots as peers and the control computer as super-peer in a similar setting. The goal of these experiments was to detect and fix different kinds of errors that may happen during the process execution and to compare the success rates of the execution with the Feedback Service (*MAPE-K+*) and without the Feedback Service (*Baseline*) [SHA17]. We used the goals presented in Listings 7.8 and 7.9. The process shown in Figure 7.29 was executed 20 times for the *Baseline* configuration and 20 times for the *MAPE-K+* setup. The robot(s) were sent to locations in the room opposite from their current location by the PROtEUS system to have relatively long distributed navigation processes. In case of an error related to the first TurtleBot (*Peer1*), the second robot (*Peer2*) was instructed to re-execute the *MoveTurtleRemote* subprocess for the *MAPE-K+* experiments. We will give a short quantitative discussion of the results without presenting detailed numbers regarding execution times and feedback loop iterations.

Results We determined four main types of errors that we are able to remedy with the help of the Feedback Service based on the aforementioned objectives. They are related to low battery levels (*Battery*) of the robots; the disconnection of the robots from the WiFi during driving and their inability to reconnect (*Disconnect*); the robots being not connected to the network or the workflow system not working from the start (*Unconnected*); and the workflow system running on the peers being overloaded with tasks and therefore unresponsive or crashing (*Overload*). The numbers of successful runs (*Success*) or erroneous runs related to the type of error are shown in Figure 7.35 for the *Baseline* experiments and in Figure 7.36 for the *MAPE-K+* experiments.

For the *Baseline* experiments 8 out of 20 runs were executed successfully, i. e., the robots reached their destinations without encountering any issues or errors. On overage, the robot was driving for approx. 43 seconds to reach its destination. In 5 cases, the battery levels dropped below a defined threshold of 10 % resulting in the robot possibly not being able to complete the driving tasks. In 3 cases, the robot

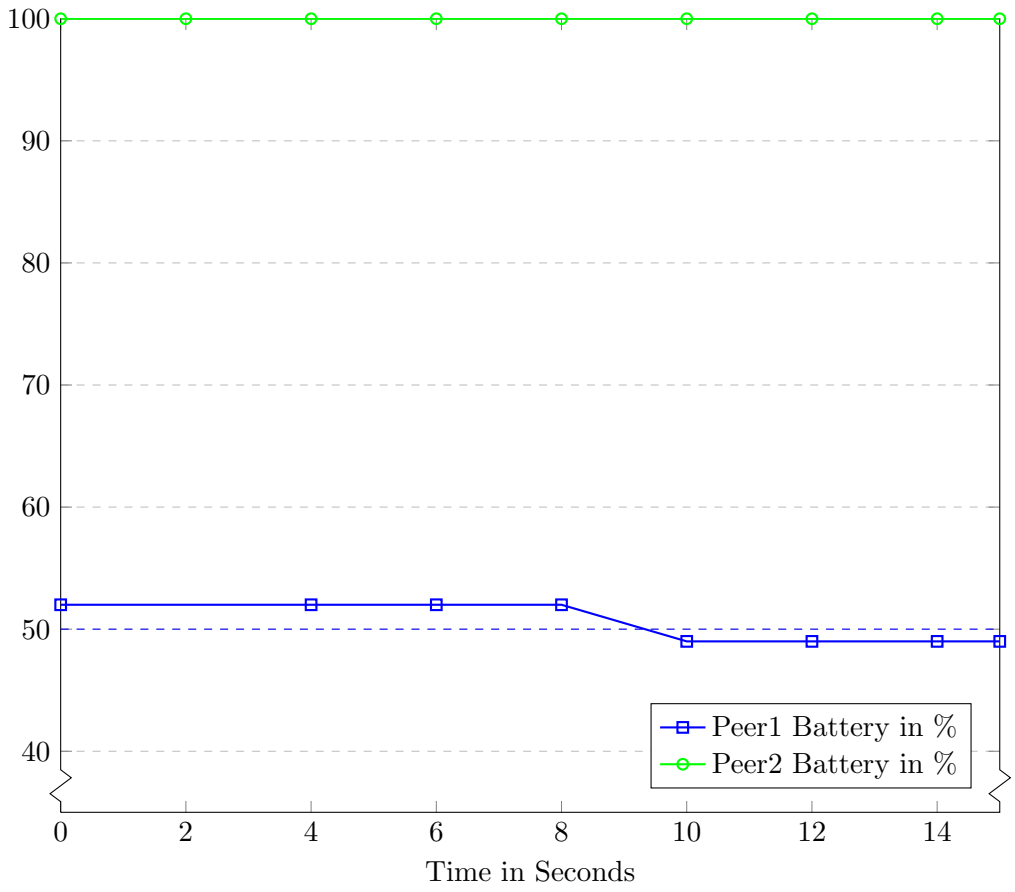


Figure 7.34.: Battery Levels of Peer1 and Peer2 over Time.

lost its WiFi connection and with that, the connection to the super-peer during driving and in 2 occasions, it was not connected to the super-peer in the first place. Another 2 issues were related to an overloaded and stalling process engine.

For the *MAPE-K+* experiments, we tried to reproduce the errors that occurred with the first TurtleBot in the *Baseline* run. The Feedback Service runs on the super-peer to check if the objectives can be fulfilled by the first peer or to choose another peer to instantiate the subprocess on. The second TurtleBot used as replacement peer has a stronger control computer with a better processor and larger battery. After the super-peer detected one of the aforementioned issues regarding the distributed process execution on the first peer, it repeated the execution on the second peer as a result of the Plan phase of the MAPE-K loops. The second robot always started right next to the first robot's starting position. The rate of successful process executions increased to 18 out of 20 runs for this configuration. We made sure that the second peer's control laptop's battery was always sufficiently charged and it was connected to the super-peer before receiving the subprocess. The laptop had more computing resources available, which prevented the process engine from being overloaded. That way the success rate of the process executions increased due to the application of the MAPE-K feedback loops. However, also the second robot can still be subject to similar errors. We encountered 2 out of 20 cases where

also the second TurtleBot got disconnected from the network during execution of the driving processes after replacing the first TurtleBot. On overage, the successful driving processes resulting from a failure with Peer1 took 53 seconds to complete with the help of Peer2. We tried to reproduce the errors that occurred for TurtleBot1 in the *Baseline* configuration approx. 8 seconds after the start of the process execution. The detection of the error, planning and redeployment on TurtleBot2 by the Feedback Service introduced an overhead of approx. 2 seconds on overage.

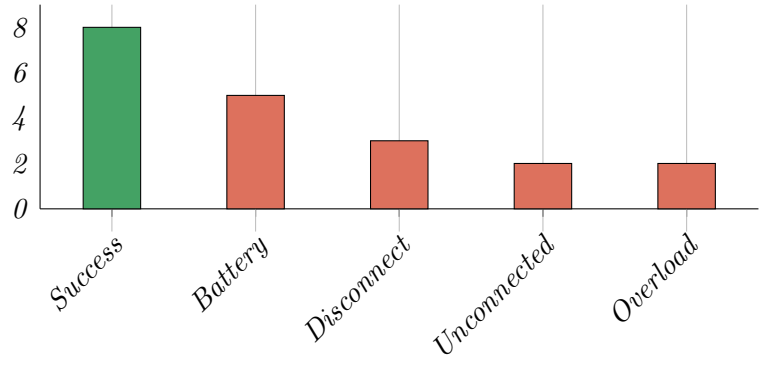


Figure 7.35.: Number of Distributed Process Executions Correlated with Successful Executions or Type of Error for Baseline Experiments.

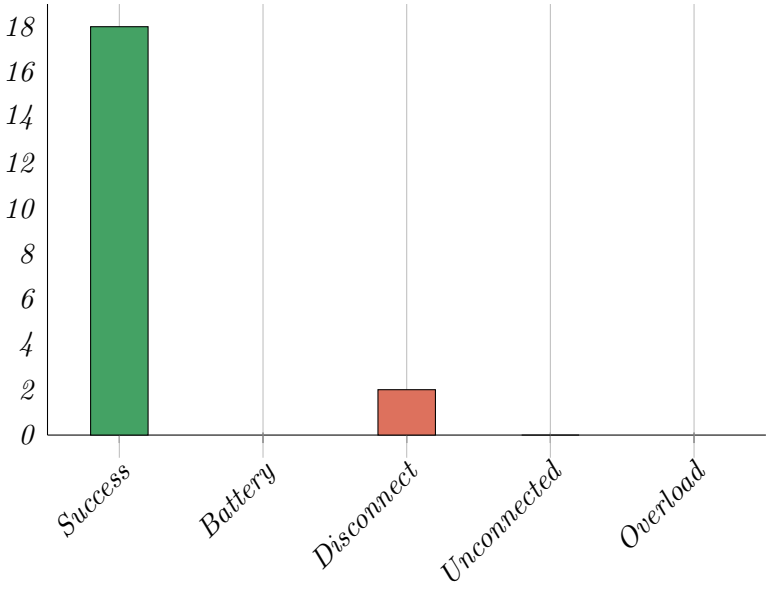


Figure 7.36.: Number of Distributed Process Executions Correlated with Successful Executions or Type of Error for MAPE-K+ Experiments.

Discussion

The results of the experiments show that the interactions between the PROtEUS WfMS and the Feedback Service also work as expected for the distributed execu-

tion of processes (Requirement *R4*). The application of feedback loops helped us to reduce the cases of unsuccessful process executions significantly. In our hybrid overlay network architecture, the super-peer runs the *D-PROtEUS* setup for distributed processes as well as the Feedback Service to monitor various defined aspects related to the process execution on the peers. By applying the MAPE-K feedback loops, we are able to detect and remedy errors related to different objectives–not only regarding cyber-physical aspects (e. g., battery levels) but also to other QoS criteria (e. g., response times or heartbeats). This shows that the Feedback Service based on the MAPE-K loop can be used as a general framework to implement self-adaptive behaviour based on various criteria (Requirement *R7*). The Feedback Service introduces a computational overhead and increases execution times, which still seem reasonable for our uses cases, though. Peers can be lightweight computers only running the basic *PROtEUS* configuration. Super-peers have to run the *D-PROtEUS* configuration and the Feedback Service additionally to perform the peer management and process distribution tasks and to enable the self-management for distributed workflows. These tasks are more resource demanding requiring more powerful computers to act as super-peers. As seen in our evaluation setup, a standard desktop computer is sufficient to run the necessary super-peer components and Feedback Service. Investigations regarding more complex distributed process execution setups involving multiple peers, super-peers and possibly several hierarchical layers are subject to future work.

The TurtleBots are perfect examples of mobile IoT devices with rather limited resources and unreliable network connectivity, which lead to the occurrence of multiple types of device related errors. The process adaptations discussed in these scenarios refer to the selection of alternative peers and the repeated execution of the entire subprocess on the new peer in case of an error. These adaptations are rather simple and often not reasonable for more complex processes in CPS. As already discussed in Section 6.2.6, actions performed by processes in the physical world cannot always be easily undone, rolled back or repeated as physical resources and objects are scarce and more vulnerable. Adaptations have to consider the types of errors and actions performed as part of the "original" process to remedy occurring issues.

7.5. Feedback Service with Legacy WfMSes

To complete the evaluation of our concepts and prototypes, we conducted some additional experiments regarding the retrofitting of existing WfMSes using the Feedback Service in combination with these "legacy" systems (Requirement *R8*) [SHA18a]. A simplified version of the *Smart Lighting* process serves as basic process to be executed by the respective WfMS and augmented with an additional call to the Feedback Service to supervise and manage the execution. The goal added to the basic process steps and used by the Feedback Service corresponds to the goal specified in Listing 7.6 with the slight change of the light levels being required to reach 600 Lux within 2 seconds for the process to be executed successfully. We deactivate the first lamp to be switched on by the basic process so that the Feedback Service has to find and dim up the second dimmer switch to reach the desired light levels.

7.5.1. Experiments

To demonstrate and evaluate the *invasive* retrofitting (cf. Section 6.12.1), we executed the process depicted in Figure 7.37 with the *PROtEUS* system in combination with the Feedback Service. A RESTful service call to the openHAB IoT middleware requesting the dimmer to be activated is sent in the "LightInvoke" process step and the WfMS uses the goal to issue a parallel call to the Feedback Service implicitly.

To evaluate the *non-invasive* retrofitting (cf. Section 6.12.1), we executed similar "LightInvoke" processes with the same goal and basic process step. The WfMSes Activiti (Version 6.0.0), Apache ODE (Version 1.3.8) and YAWL engine (Version 4.2) were used to execute the respective processes shown in Section 6.12.2 consisting of an invocation of the basic process activity to switch on the light and an explicit call to the Feedback Service with the goal as input parameter in parallel. All WfMSes were running on the control computer. Where necessary, we used an Apache Tomcat[17] (Version 9.0.7) as server for the respective web applications. The particular processes and additional services to delegate the service calls to the middleware and Feedback Service are available on GitHub[18].

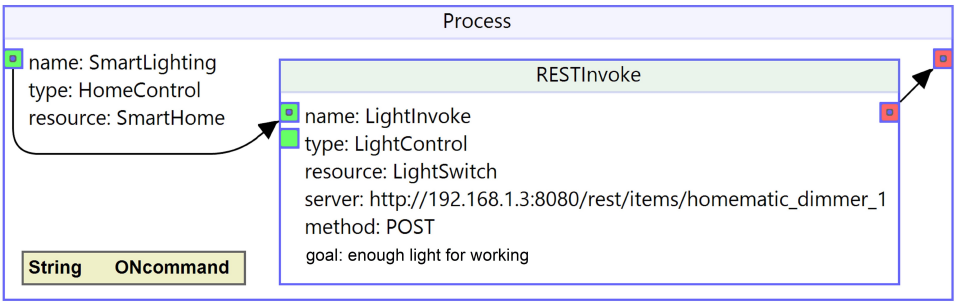


Figure 7.37.: *Retrofitted* Smart Lighting Process for *PROtEUS*.

7.5.2. Results

The execution times for the specific "LightInvoke" activity and the overall process including the execution of the MAPE-K loops can be found in Table 7.16. The execution of the basic *LightInvoke* process step (cf. Figure 7.37) as an asynchronous REST service call to the middleware took 193 milliseconds with **PROtEUS**. The parallel invocation, error detection and compensation of the broken light switch by the Feedback Service dimming up the second light source stepwise from 85 Lux to 657 Lux took approx. 24 seconds. The execution of the basic BPMN *LightInvoke* step (cf. Figure 6.12) took 459 milliseconds with **Activiti**. The parallel invocation of the Feedback Service dimming up the second light stepwise from 89 Lux to 766 Lux took approx. 22 seconds. The execution of the basic YAWL *LightInvoke* step (cf. Figure 6.13) took 171 milliseconds with the **YAWL Engine** and the parallel invocation of the Feedback Service dimming up the second light stepwise from 81 Lux to 685 Lux took approx. 22 seconds. The execution of the basic WS-BPEL *LightInvoke* step (cf. Figure 6.14) took 98 milliseconds with **Apache ODE**. The

[17] `http://tomcat.apache.org/`
[18] `https://github.com/IoTUDresden/fbs-retrofit`

parallel invocation of the Feedback Service dimming up the second light stepwise from 93 Lux to 639 Lux took approx. 24 seconds.

Table 7.16.: Execution Times of the Basic "LightInvoke" Activity and the Overall *SmartLighting* Process for the Investigated WfMSes.

WfMS	LightInvoke (in ms)	Process (in s)
PROtEUS	193	24
Activiti	459	22
YAWL Engine	171	22
Apache ODE	98	24

7.5.3. Discussion

The results of the retrofitting experiments show that the proposed retrofitting processes are feasible to be applied for adding autonomous capabilities to the investigated WfMSes (Requirement *R8*). We were able to try to switch on the light, detect the malfunction of the light switch and find an alternative light source to be dimmed up successfully with all WfMSes. The overall process execution times depend on different physical context factors (e. g., the initial light levels and the detection rate of the light sensors), which is why a comparison of these execution times is not reasonable. As already concluded from the previous experiments, activities executed in the physical world have a much stronger impact on the overall execution times than the virtual computations. The comparison of the execution times of the basic "LightInvoke" process activity either invoking the middleware directly via a REST service or via intermediate services/applications shows good performance benchmarks for *PROtEUS* with execution times being in the same order of magnitude as the execution times of other WfMSes.

7.5.4. Summary

The experiments described in the previous sections and their results show that the *PROtEUS* system in the basic configuration as well as in the *D-PROtEUS* setup is capable of executing CPS workflows in the context of a smart home with respect to the Requirements *R1–R4* identified in Section 2.6 (*Proof of Concept*). With the case study experiments, we were able to show that the processing of complex sensor event streams and triggering of high-level process events; the dynamic selection of services based on the IoT devices' capabilities and context; the ubiquitous interaction with and by humans; and the distribution of process fragments among execution peers and managing super-peers can be modelled and executed with the *PROtEUS* WfMS and its associated tools (e. g., the process editor, the process manager and the human task manager app described in Sections 4.8.1 and 5.6). More complex IoT devices consisting of multiple sensors and actuators (here: service robots) can also be integrated as resources into CPS workflows by applying capability based abstractions as shown with the *DROiT API* in [SSAS15] and Section 4.3.3. The investigated example processes represent rather simple workflows in the smart home with only few parallel activities and instances to be executed. The case studies have

to be extended to a larger scale as part of future work to evaluate the feasibility, scalability and correct interactions of the PRotEUS WfMS, Feedback Service and their individual components. Despite the processes that have been investigated being relatively simple, a large number of software components and complex configurations were necessary to implement a running WfMS including the required monitoring and logging infrastructure.

The interactions of the CPS workflows with the physical world require additional means for handling unanticipated errors due to the IoT devices being more unreliable and physical obstacles and other new context factors influencing the outcome of the process executions. These new error sources require feedback from additional data sources about the process instance execution to check for *Cyber-physical Consistency* and detect and repair these errors (Requirements *R5–R7*). From the experiments we see that these requirements can be covered by the *Feedback Service* based implementation of the MAPE-K feedback loops extending the "regular" process execution. By using additional externals sensors and other information sources, we are able to define and check criteria/rules for the successful execution (*satisfied condition*) of a process step or for a potential error (*compensation condition*). The experiments show that this correlation also works as expected–to just verify the successful execution but also to adapt the process in case of errors–and we are able to increase the rate of successful process execution with the help of the Feedback Service compared to only using the basic WfMS. The Feedback Service is able to detect violations with respect to the defined criteria regarding cyber-physical consistency (Requirement *R5*) but also more general criteria (Requirement *R7*). Simple adaptations derived and executed by the Feedback Service facilitate the cyber-physical synchronization and resolving of errors (Requirements *R5* and *R6*) as well as more general adjustments of the WfMS at runtime with respect to other self-* properties besides self-healing (Requirement *R7*). The adaptation strategies are currently limited to exchanging the process resources and repeating the respective process activities, which are rather simple adaptations. More complex strategies can be defined based on new compensation queries to be added to the compensation repository of the Feedback Service to specify how to select new process resources or adapt the process instance. We were also able to proof the feasibility of our proposed retrofitting processes for existing WfMSes (Requirement *R8*) to add self-* capabilities to these systems by using the Feedback Service. Similar to the basic PROtEUS system, the Feedback Service shows a reasonable performance and execution times for the smart home use case when executing the MAPE-K loops. The execution times of the individual phases are also in the order of 100 milliseconds or less. The major contributors to the overall MAPE-K executions are again the relatively slow sensors measuring and reporting data from the physical environment. The overhead from communicating with the Feedback Service as an external service can be neglected at this point with a more integrated solution providing even better performance. In general, with our smart home case study and experiments, we are able to address all requirements *R1–R8* and show their fulfilment by our concepts and prototypes.

The PROtEUS system in its various configurations and in combination with the Feedback Service shows reasonable execution times for virtual and physical activities in the smart home context. The durations of virtual computations to execute simple process logics, send service requests or retrieve data from the SAL are in the

order of 100 milliseconds or less, which makes the system also capable of executing more crucial tasks demanding (soft) real-time like behaviour. An evaluation of the system's performance and its scalability in an extended case study with respect to the execution of multiple and more complex process instances remains subject to future work. The experiments show that the interactions with the physical world take significantly longer due to the physical processes involving humans, objects and the environment being much slower than the computations executed by the processor of the WfMS's hosting computer. These long running physical process activities also show the importance for supporting event-driven (asynchronous) communication and behaviour within the workflows and WfMS (Requirement *R1*) as the process execution often has to wait for activities to finish–without knowing their actual execution times–in a non-blocking way. As we focus on real-world experiments with actual data from the physical world to verify the behaviour of the WfMS in specific case studies and to identify new sources of potential errors, a performance comparison with other approaches and WfMSes is not reasonable due to the long running physical activities that are the major contributors to the overall execution times. Despite not providing any guarantees or mechanisms for real-time executions, we are able to define time-dependant behaviour and constraints using EPL statements for the triggering of events, and using goals to specify deadlines for the successful execution of process activities and to detect their violations with the Feedback Service during or after execution.

7.6. Qualitative Discussion of Requirements and Additional CPS Aspects

Following the detailed quantitative analysis of the experiments regarding CPS workflows in the smart home case study, we use the results to provide a qualitative discussion of the fulfilment of the requirements *R1–R8* as well as of additional CPS aspects–namely *Real-time, Security and Safety*–in the subsequent sections.

7.6.1. R1: Abstraction and Processing of Complex Sensor Events

Reactive, event-driven behaviour is identified as an important key property of CPS that needs to be supported by a WfMS. As shown with the experiments, process activities interacting with the physical world are relatively long running and should be executed asynchronously, i. e., react to events emitted by the respective services to signal relevant changes in the service execution. We introduce a special process step (*TriggeredEvent*) to define the occurrence of a particular event on the process level that triggers the activation of the connected process steps. EPL patterns provide an expressive syntax to define event patterns–from the occurrence of a single low-level event to complex fusions of multiple low-level events over time including arithmetic operations to preprocess the events. With that, we can define event patterns related to a wide range of physical and digital event sources (sensors) with arbitrary complexity, granularity and quantity of lower level sources. The CEP engine proved itself to be an indispensable high-performance component of the WfMS to process complex streams of sensor data and detect relevant patterns in these streams produced by the CPS entities. The specification of EPL patterns is a simple means for

enabling context recognition and context-dependant behaviour in workflows. Based on detected sensor patterns (contexts), the corresponding process-level events can trigger the activation of different branches within a process and thereby add a basic level of context awareness to the processes. Our exemplary processes rely on relatively simple event streams with sensor events from single sources occurring only a few times per second. Simple event handling mechanisms suffice at this point to detect certain patterns within the event streams. However, market researchers predict multiple billion IoT devices–sensors and actuators–being sold and deployed in almost every area of everyday life in the next decade [RW15]. The increasing number of sensors and other event sources requires a more scalable form of event processing–CEP being a suitable candidate to achieve the necessary performance [Luc02]. To increase the robustness of the event detection and activation on the business process level, multiple sensors sources and redundant sensors and types of sensors can be considered in EPL statements and processed by the CEP engine (*Sensor Fusion* [BI98]). Regarding the granularity of events, we support the processing of events of arbitrary complexity–from low-level sensor events referring to specific sensor instances to type-level specifications and aggregated events provided by more sophisticated software-based event sources. The aggregation of events and derivation of higher level events is left to the dedicated software components acting as event provider for our CEP engine. One prerequisite for using these events is their containment in the knowledge base so that EPL patterns can refer to them. With *TriggeringEvents* we have a simple means of defining events and corresponding payloads on the process level that can be injected into the event cloud and considered by the CEP engine for other processes.

Regarding the aspect of *Instance Correlation*, i. e., correlating events with each other as well as with specific workflow instances and vice versa, our experiments show that by linking sensor events and the execution of event-related process steps of a workflow instance, we are able to relate the execution with the respective low-level events. The EPL statements provide a mechanism to specify these correlations for an arbitrary number and granularity of sensors and the corresponding process steps in the process model. By investigating the specific instance execution logs and sensor history, a correlation of the appearance of specific events with the execution of the related process instance activities is possible for single process instances. Vice versa, *Goals* can be used to link the process instance executions to their effects on the physical environment and with that, the appearance of specific (sensor) events as consequences of the instance executions. The objectives associated with a process step define expected changes within specific sensor values–event sources in general–and error criteria related to these events. The main purpose of the MAPE-K feedback loop is to monitor the relevant event data and correlate the process instance execution with these data to detect success or error of the execution. With our experiments, we were able to show how to establish these links based on the feedback loops. Within the scope of this thesis, we only investigate the execution of single process instances isolated from each other. Process models are only instantiated and executed once, which is why the aspect of correlating the occurrence of multiple (cyber-physical) events with the execution of multiple process instances (e. g., by linking the process instance to the corresponding event instance via the instance identifiers) has to be further investigated in larger scale experiments. Cur-

rently, the WfMS does not address and support this correlation explicitly as the boarders of individual process instances may not always be clear in CPS. The interactions of multiple process instances and feedback loops with the physical world may also lead to additional issues regarding the access to limited physical resources and appearance of concurrent or parallel events.

The specification of EPL statements in our approach is currently not supported by convenient tools, which is why the definition of EPL patterns is rather complicated and requires deep knowledge about the corresponding syntax as well as the underlying sensor and domain model. Only sensors described by the global sensor and event model, and connected via their respective software adapters can be considered during the operation of the CEP engine. The EPL statements currently have to be specified at design time and are therefore rather static. Besides directly addressing specific sensor instances as shown in our example processes, types of sensors can also be considered to cope with the dynamic availability of resources. The process designer has to specify these events for the particular processes in accordance with the respective event data model. The learning of event patterns and EPL statements or the automatic extraction of specific event patterns from business processes as described in [WZG+14] can be first steps to further increase the flexibility of our approach regarding the pattern-based specification and detection of events on the business process level, as well as to facilitate the automatic discovery of business processes from event streams [vZvDvdA18].

Advantages:

- Event-driven behaviour is enabled on the business process level.
- Event abstractions in processes on arbitrary levels and granularity w. r. t. the event sources
- Complex arithmetic and time-related expressions/patterns with EPL statements referring to multiple sensor and event sources
- Fast event stream processing with CEP engine
- Sensor fusion and context recognition in processes

Limitations:

- EPL statements are rather static and complex to define.
- Event models have to be known and event adapters developed.
- Limited correlation of the appearance of (sensor) events with the execution of process instances.

7.6.2. R2: Integration and Dynamic Selection of Resources

By relying on services for accessing the functionality of all physical and virtual CPS entities, we followed the suggestions of the IoT reference model described in Section 2.4.4. Services provide a unified abstraction for communication with the heterogeneous devices and virtual entities of the CPS that can be used as process resources and invoked by the WfMS based on standard protocols and interfaces. The

WfMS supports an extensible set of IoT service invocations using standard or proprietary protocols on the business process level. The description of the CPS devices, their contexts, functionality, capabilities and associated services within an ontology allows us to specify required functionality and constraints that the respective process resource needs to execute a particular process step. The semantic queries together with the SAL provide us with means for finding and invoking suitable services and resources at runtime based on their availability, properties, functionality and context. The experiments show that the dynamic selection of process resources, retrieval of parameters and sensor values (*SemanticSelect*), the checking of conditions related to sensor values (*SemanticAsk*), and the retrieval and invocation of actuator functionalities (*SemanticCommand*) work well for CPS devices according to their type, functionality and context in the smart home. More complex compounds of sensors and actuators can be abstracted and integrated following a generalization of the capability-based approach we presented for the service robots (cf. Section A).

Using IoT services as the basic abstraction and communication approach to invoke functionality provided by CPS entities remotely is a feasible way of accessing the heterogeneous devices in a platform-independent way. However, it also introduces additional overhead as the respective control software needs to be augmented by a dedicated server component hosted on the particular device or its control computer (cf. Figure 2.15). We have seen that the underlying infrastructure for providing access to the specific devices also plays an important role when designing the WfMS and CPS architecture in general. In our examples, the openHAB middleware is the central component to host services for accessing the sensor and actuator functionality. Other services can also be invoked via their specific URIs and the middleware can be used in a decentralized setting. However, the SAL currently relies on a central knowledge base containing the semantic descriptions of the entities and their services. The underlying ontology is rather static and cannot be modified with new elements at runtime. Similar to the EPL statements, the corresponding SPARQL queries are also static and complex to define as part of the semantic process steps and require deep knowledge regarding the specific syntax and domain model (ontology). Compared to more traditional approaches for service discovery (e. g., using *UDDI* [CDK+02]), we provide an advanced and more sophisticated mechanism for finding services at runtime based on semantic data, which can be further enhanced by using inference to discover new process resources from implicit knowledge [DBBM11]. Many other approaches for CPS and IoT service discovery exist [BK17, TFR17, DBBM11, SZY18], but they were not applied in the context of business processes and process resource discovery with semantic contextual data. We currently use the knowledge base and ontology as a sophisticated data model and storage for semantic search, not exploiting the semantic information to infer new knowledge. The performance and feasibility of these mechanisms for service discovery and invocation at runtime have to be further evaluated in the context of CPS and IoT, which usually require a certain level of responsiveness that reasoning algorithms and large ontologies may not be able to achieve. An advancement of our semantics-based process resource discovery and assignment approach in the context of IoT processes using goals founded on the TROPOS approach [BPG+04], roles and reasoning can be found in [Hub18].

Advantages:

- Unified abstraction of CPS resources as IoT services
- Complex actuators and sensors can be used as process participants.
- Extensible support of standard and proprietary service protocols
- Service discovery at runtime based on properties, functionalities, capabilities and context of CPS entities

Limitations:

- Encapsulations as service necessary
- Semantic queries are rather static and complex to define.
- Central knowledge base with complex, rather static ontology
- No application of semantic reasoning techniques

7.6.3. R3: Ubiquitous Interaction with Humans

From the use cases and experiments, we have seen that the interaction with users as part of the process execution as well as for managing the process execution is an important requirement despite the overall goal of increasing the level of automation within the CPS with the help of workflows. With human tasks, we can explicitly specify manual activities and tasks as process steps that need to be executed by humans on the business process level. These information can then be used to create simple user interfaces based on UI components available from the underlying platform to specify what to do, display input data and provide forms for entering output data. The dynamic generation of more complex, more generic and sophisticated user interfaces is out of scope of this work. The publish/subscribe communication facilitates the loose coupling of arbitrary clients with the WfMS to work on human tasks, to monitor the workflow execution or to control processes on the corresponding channels (*topics*). Our prototypes feature stationary but also mobile client applications and tools to model and manage processes. The modelling and monitoring tools are rather complex to be used but mature implementations as they are founded on well-established user interface frameworks. These applications enable a deep inspection and debugging of the processes and process instances to support the succeeding redesign and process improvement phases. Especially the mobility and location aspects become more and more important with the development of CPS towards ubiquitous systems. Our mobile applications are the first steps to support a more end-user friendly management and creation of workflows and CPS in general, which can be best experienced with the *HoloFlows* mixed reality application (cf. Section 4.8.2). The mobile and augmented reality process management apps as well as stationary tabletop application are working prototypes that still have to be improved and matured towards useful process management applications.

Despite developing interactive applications with more end-user friendly user interfaces, we do not focus on explicitly addressing specific characteristics or requirements of end-users or reaching a high level of user experience in general. The process modelling and management tools are still rather complicated and require knowledge

about the specific domains as well as about the workflows themselves. Our focus is on providing a set of sophisticated applications for workflow and CPS management. Future research and developments should increase the usability of these applications by simplifying certain modelling and management aspects and following a more user-centred design process [AMkP04]. In addition, several modelling and configuration tasks can be automated to reduce the complexity of the overall CPS management processes (cf. Section 4.8.3). This could include the automatic discovery and configuration of known and unknown CPS devices as process resources, up to the automated learning and generation of complete CPS workflows based on activity recognition.

Advantages:

- Seamless integration of human interactions/manual tasks into processes
- Simple dynamic user interfaces for human tasks
- Flexible loose coupling of end-user devices with the WfMS
- Comprehensive means and tools for process management and modelling
- End-user friendly multi-modal applications for process management

Limitations:

- No explicit addressing of human capabilities and context
- No complex user interface generation
- No explicit focus on usability
- Complex modelling tools require workflow, CPS and domain knowledge.

7.6.4. R4: Distributed Process Execution

The hierarchical overlay (peer–super-peer) network infrastructure that we base the distributed execution of processes on, provides a scalable architecture to manage CPS devices involved in the process execution. Compared to a fully decentralized system of systems, this approach reduces communication and coordination overhead due to central management entities (super-peers) running the *D-PROtEUS* configuration of the WfMS [YGM01]. With our approach, the hierarchies available in the typical network infrastructures for Fog Computing can be used to establish a distributed process execution infrastructure of peers, super-peers and higher order super-peers. We support the distribution of process steps on the level of individual tasks, subprocesses or entire processes by relying on the subcontracting mechanism, i. e., the particular process fragment and instance information are transferred to the designated remote PROtEUS system and then executed locally (instance migration [ZHKL10])–possibly on specialized edge computing devices. That way, also a certain level of data security can be reached as only required process data is transferred to the respective peer. The peers do not depend on a constant connection to the super-peers or to the network in general and are therefore able to execute their assigned subprocesses autonomously.

The actual distribution of process tasks to remote WfMS is currently done in a relatively simple manner relying on the explicitly specified resource URI property

of the specific process step. More sophisticated means for automating this assignment (e. g., based on a peer's capabilities, available computing resources, proximity or other context factors) could be investigated as part of future work. Similarly, the assignment of the roles of peers and super-peers is currently done manually, but can also be automated based on the aforementioned criteria. In general, super-peers require more resources for managing the network infrastructure and a more reliable network connection. Peers can be resource-constraint and more specialized. The management of the network infrastructure on the super-peer level is currently out of scope of our work but also needs to be considered in future developments (e. g., the handling of super-peer failures). The implementation of the subcontracting mechanism for transferring the respective process instance and its current state to a peer is also relatively simple. In [BDH+11] Barkhordarian et al. discuss possible issues with migrating business process instances to other execution systems. These aspects should be studied and considered in future developments to increase the robustness of our system. In addition, we do not implement any mechanisms for ensuring transaction safety when transferring and executing subprocesses on remote peers. The mechanisms for handling long running transactions in pervasive workflows proposed by Montagut et al. in [MMG08] are reasonable extensions of our system. All in all, we provide a basic implementation of a distributed process execution system that can be improved in future developments and tested in larger scale SoS.

Advantages:

- Distributed execution on the level of tasks, subprocesses or processes
- Hierarchical network enables locality of process execution and scalability.
- Data security through subcontracting and instance migration
- Offline execution and specialized task execution on specific devices

Limitations:

- Simple distribution of process tasks based on resource URI
- Simple assignment of peer and super-peer roles
- No communication among super-peers
- No failure handling w. r. t. super-peers
- No consideration of long running asynchronous transactions

7.6.5. R5: Cyber-physical Synchronization

With CPS workflows adding the new dimension of influencing the *physical* world to business processes, the linking of the workflow execution to its real-world effects becomes a key criterion for implementing a suitable WfMS for CPS. By relying on the MAPE-K control loop and its implementation within the Feedback Service, we are able to create this link with the help of additional physical information sources measuring physical context factors (sensors) that are considered in order to confirm the success of the process execution and thereby the successful cyber-physical synchronization, or to determine a possible error leading to an inconsistent state.

Both criteria (success, failure) are defined in a more declarative manner within *Objectives* that are part of an overall *Goal* for the execution of individual process step instances. Goals define *What* is expected to happen when executing the respective process step, but not *How* it is supposed to happen. This adds more flexibility to the process execution and reduces complexity as error cases and failure handling do not have to be specified explicitly as part of a process model. Goals may contain objectives regarding multiple context factors from various sensors and complex evaluation criteria–also considering time-related aspects–relevant for determining *Cyber-physical Consistency* or the need for synchronization due to errors or deviations. Our investigations show that, besides physical context factors, also arbitrary other relevant context factors can be considered within the MAPE-K feedback loops to verify the execution of individual process steps or to define more general constraints for the process execution.

The goals and objectives used to specify success and error criteria are currently relatively complex to define and require deep knowledge about the used sensor models, domain-dependant workflow effects and syntax. The linking of the effects of the workflow execution to their physical outcome is not always straightforward and requires additional domain knowledge. As workflows are abstract virtual concepts that do not have an explicit physical counterpart, changes within the physical environment and related to the state of objects, CPS devices, things, humans or other relevant entities and criteria have to be considered to realize the aspect of cyber-physical synchronization for workflows. The specification of goals has to be simplified and supported by the IDE in a more sophisticated way. Additional checking mechanisms have to be considered to identify possibly conflicting and concurrent goals also considering the semantics of the associated process step. The relevant context factors to be considered within the MAPE-K loops have to be part of the sensor model and integrated as data sources. Goals are currently rather static, i. e., they have to be defined at design time and cannot be modified at runtime. Adding the capability of dynamic adaptations and meta-adaptations [PRK+14] to the goal management as well as automated learning of goals are part of future work.

Advantages:

- Linking of workflow executions to their physical effect and vice versa with the help of sensors
- Synchronization criteria defined in goals (success or failure) for Cyber-physical Consistency–more flexible and less complex than explicit definition
- Complex expressions in goals possible, including multiple objectives, sensors and time-dependant criteria
- Non-CPS related workflow effects/conditions can also be specified.
- MAPE-K loop and Feedback Service as suitable framework for cyber-physical synchronization

Limitations:

- Knowledge base has to contain sensor model, sensors have to be integrated and constantly updated.

- Goals are rather static and complex to define.
- No explicit physical representation of a CPS workflow possible
- Linking of workflow execution to physical effects not always straightforward, requires domain knowledge

7.6.6. R6: Handling of Cyber-physical Errors

Based on the goal definitions and additional environmental sensors as well as state changes within relevant CPS entities, we are able to verify the expected outcome of the workflow execution (see *satisfied condition* in Section 4.5) or to detect errors in case of deviations from and undesired states of the physical world (see *compensation condition* in Section 4.5). As shown within the evaluation and discussions, this approach is feasible for detecting cyber-physical mismatches and finding compensation strategies automatically within the MAPE-K loops. The investigation of other formalisms to specify these goals on a more abstract level, e. g., as proposed in [KK12] or with the TROPOS methodology [BPG+04] remains subject to future work. In case of an error, the Feedback Service can detect the type and quantity of the mismatch, which is the basis for finding a compensation strategy. Currently, the main strategy for deriving compensation actions is to look for a suitable replacement resource as defined within custom compensation queries to repeat the execution of the respective process step on and to check again for successful execution within the MAPE-K loop. These compensation strategies/adaptations are rather simple but suitable for our error cases as the process resources (CPS devices) are the main new source for errors in CPS workflows. However, a simple repetition of the same process step execution is often not feasible due to more complex and possibly concurrent processes happening in the physical world. A more sophisticated classification of process tasks, errors and corresponding rollback/undo/compensation actions is required to define suitable compensation queries, which are very complex to specify, or to automatically derive compensation strategies. As compensations and rollbacks cannot be discussed in their entire complexity within the context of this thesis, we leave this as an open task for future work. With the help of *Consistency Levels*, we are able to influence the level of precision and consistency required for the successful execution of a process step, which usually results in execution times and number of feedback loop iterations that are proportional to the desired consistency level.

The detection of errors is currently done based on simple rules/criteria defined within a goal. These criteria may refer to several physical context factors that have to be reached, also with respect to certain time frames. The error detection mechanism could be augmented in future work, e. g., by applying stream-based process mining to check for conformance from process events [vZvDvdA18]. Currently, we are not always able to detect the actual error source, but only that a desired state cannot be reached, e. g., a process is supposed to switch on various lights in the room but the expected light levels are not reached due to a broken light bulb. The goals and processes have to be specified in more detail to address this issue. Our approach enables the self-healing with respect to cyber-physical errors by adapting the process and thereby trying to synchronize the actual physical state with the assumed state (target value) defined in an objective. This assumes that the workflow designer is able to specify the outcome of the specific process step exe-

cution. More sophisticated process adaptation and error compensation strategies have to consider the type of the basic process activity and failure that occurred to find suitable remedies. We also do not explicitly consider concurrent processes and process instances being executed in the physical world within our investigations.

Advantages:

- Error detection based on additional sensor data from physical environment
- Compensation of physical errors in MAPE-K loop
- Scalable precision and consistency through goal definitions

Limitations:

- Compensation strategies rather simple
- Localization of errors not always possible
- Classification of physical errors and remedies necessary
- No explicit handling of concurrency

7.6.7. R7: Self-management Capabilities

The elaborations regarding the handling of cyber-physical errors are also relevant for discussing the self-management capabilities of the CPS WfMS proposed in our work as self-management is a generalization of self-healing and other self-* properties (cf. Section 2.4.5). Goals can be used to define success and error criteria regarding arbitrary factors related to the process execution (e. g., QoS, KPIs or cyber-physical context values). These factors have to be known in advance by the workflow designer, be part of the sensor model and be connected to the Feedback Service via dedicated *Monitoring Agents* (cf. Section 6.2.1). Currently, goals are static and cannot be adapted at runtime. Future developments could comprise the implementation of more dynamic goals and meta-adaptations, e. g., by regarding the phases of the MAPE-K loop as individual process steps and specifying for goals for applying the MAPE-K loop again to the individual phases.

The Feedback Service provides a generic and extensible implementation of the MAPE-K control loop, which proves to be a suitable framework for enabling self-awareness and self-management of software systems [MSW16]. Arbitrary data from additional sources can be considered within the Analyser component of the Feedback Service and also considered when deriving suitable compensation strategies in the Planning phase. The compensation repository may contain arbitrary compensation queries and compensation actions to find suitable adaptations for the process or process resources to reach the desired goal. This does not only relate to the self-healing that we have shown with respect to cyber-physical processes and distributed processes, but also to the implementation of other self-* properties (e. g., self-configuration or self-optimization). Our examples related to cyber-physical processes and distributed processes show very simple adaptations of the underlying process instances or process resources (ad-hoc adaptations) in the form of repeated process step executions. These strategies are not always feasible, especially when repeating subprocesses containing physical actions that cannot be easily

repeated to due constraints of physical resources. As already mentioned, a more sophisticated classification of process tasks and corresponding rollbacks in case of errors are necessary to implement self-managed CPS workflows with transaction safety [MM05]. To extend the compensation strategies and compensation queries as well as to add new monitoring agents requires developers with in depth knowledge of the underlying data models, syntax and interfaces of the Feedback Service. Complementary to the proposed adaptations regarding the process resources and process execution, structural modifications of the corresponding process instance and process model should be considered in future developments. The investigation of concepts regarding the implementation of flexible processes within the context of the *ADEPT* project [DR09] to deal with errors and unanticipated situations in CPS have to be part of future developments and improvements of the PROtEUS WfMS.

The MAPE-K feedback loop can be regarded as a process itself, which is executed as part of the managed process step execution. The explicit modelling of each phase of the MAPE-K loop including data analysis as well as planning of compensation strategies as dedicated process steps is not feasible though, due to an increasing complexity of the respective process models and executions. The MAPE-K feedback process should be transparent to the process designer and part of the integrated functionality of the execution engine, only using the goals defined by the modeller for executing the control loops and adapting the processes in case of exceptions.

Advantages:

- Goals define criteria for success or undesired behaviour during process executions w. r. t. to arbitrary factors.
- Self-* capabilities and self-awareness of workflow executions via MAPE-K framework and additional data sources
- Self-management (self-healing) for distributed workflows
- Flexible and exchangeable analysis and compensation/adaptation strategies

Limitations:

- Workflow related effects (for success, failure) have to be known in advance.
- Rather simple workflow adaptations
- Classification of "undesired" situations and corresponding remedies is required.
- Compensation strategies complex to define
- No meta-adaptations

7.6.8. R8: Retrofitting Framework for Workflow Management Systems

The realization of the Deming cycle via the MAPE-K feedback loop proved itself to be a suitable framework to add the capability of self-management to enable self-* capabilities for processes to the PROtEUS CPS WfMS. With the Feedback Service, we have a generic software component represented as a web service that implements the MAPE-K control loop. Almost all of the existing WfMSes from industry and academia are capable of invoking web services, which is why the coupling of an

existing WfMS with the Feedback Service to retrofit the "legacy" system with self-management capabilities is relatively straightforward. Either modifications to the basic processes are necessary to invoke the Feedback Service containing the goals for the respective process step as parameters in parallel (*Non-invasive Retrofitting*); or the process metamodel has to be extended with the goal specification for a process step, and the execution engine has to place a service call to the Feedback Service in parallel in the background (*Invasive Retrofitting*). Consistency Style Sheets facilitate the separation of self-management aspects from the "original" process definition and reduce the need for modifying the processes or WfMS even more.

Our proposal of loosely coupling the Feedback Service with the existing "legacy" WfMSes currently relies on a service-oriented approach using RESTful web services. As many existing WfMSes from the business process domain rely on other protocols (mostly SOAP) to communicate with external services, we often have to provide mediation web services to be deployed locally within the WfMS and to then call the Feedback Service via REST. The extension of the Feedback Service with a SOAP interface would be a simple solution for this issue. The Feedback Service can also be integrated into the WfMS more tightly as a new software component communicating with the other internal components as specified by the legacy workflow system's component/software architecture. In order to implement self-managed workflows for *CPS* or *IoT*, an additional component, information system or service-based application (e. g., IoT middleware) has to be available besides the Feedback Service to provide access from the WfMS to sensors and actuators via their respective services.

Advantages:

- MAPE-K as general framework to add self-* capabilities to existing WfMSes via the Feedback Service
- Only minor modifications to processes or WfMSes necessary
- Separation of concerns with Consistency Style Sheets, "legacy" processes do not have to be modified.

Limitations:

- Approach currently relies on service-oriented WfMSes.
- Additional service (Feedback Service) and optional mediation services have to be available, which introduces overhead.
- New data sources have to be available via additional software components.

7.6.9. Time-dependant Behaviour

Despite being out of focus of our work, *Time* plays an important role in designing and implementing CPS control systems. The results of our case study show that the interaction with the physical world usually comprises long-running asynchronous interactions, which need to be monitored and verified with additional data. Within our realization of CPS workflows, we are able to specify and check time-related execution criteria and aspects at various points. First, *Escalation Ports* can be used to define timeouts regarding the execution of process steps. In case this timeout is reached, the "regular" process execution is halted and the process branch belonging to the

escalation port is activated. Our *Emergency* scenario process shows the application of this mechanism to deal with an unconscious human being (cf. Section 7.3.2). The respective timeout has to be specified by the process modeller.

With CEP, we include a high-performance component for processing of complex sensor and event streams. EPL statements provide means to define patterns within the low-level event streams that also relate to specific time frames or points in time to trigger a higher level event. *TriggeredEvents* specified on the business process level and leading to the activation of their following process steps can be used to model preconditions for the execution of subsequent process steps similar to EPCs (i. e., a certain event has to be triggered before the respective process step can be executed). *Goals* on the other hand, may contain objectives that need to be fulfilled during or after the execution (postcondition) of a specific process step. As shown with our experiments, the objectives belonging to a process step can also contain time-related criteria defining the success or failure of the process step execution.

The experiments conducted in the context of a smart home show reasonable execution times for a CPS WfMS. The purely virtual computations rely on the performance of the underlying (Java) virtual machine and are in the order of a few milliseconds indicating the capability of achieving soft real-time. The execution of process activities that interact with the real world usually take much longer due to the nature of the physical world and physical entities acting and reacting much slower than a computer. Our CPS WfMS contains components that allow for a fast processing of sensor data and the fast execution of process tasks. However, we are not able to guarantee and implement real-time behaviour regarding the execution of CPS workflows. None of the software components chosen for PROtEUS and its associated services has an explicit focus on supporting hard real-time. With the help of the Feedback Service, we are at least able to detect the violation of real-time related aspects by evaluating time-dependant goals and objectives belonging to a process step in the MAPE-K feedback loops. In order to evaluate the performance and feasibility of our CPS WfMS in other domains, more extensive experiments and discussions of the respective time-related requirements and possibly new software components are necessary. These investigations have to be part of future work.

Advantages:

- Definition of timeouts in *Escalation Ports* for synchronous and asynchronous process activities possible
- Definition of time-dependant behaviour and preconditions for process step executions based on events in EPL statements
- Definition and check of time-dependant success/failure criteria for workflow executions in goals
- Violations of time constraints can be detected.
- WfMS-related components show execution times suitable for achieving soft real-time.

Limitations:

- No strict real-time and real-time guarantees
- No explicit focus on time-related aspects

7.6.10. Security and Safety

Similar to the consideration of time-dependant behaviour, we have not explicitly integrated mechanisms regarding safety and security in the CPS WfMS. Safety-critical actions to be executed as part of a cyber-physical process have to be handled and implemented by the respective device closer to the hardware level (cf. Section 2.5.1). However, not all safety related constraints can be built in to the respective devices as in CPS, the devices interact with other devices, the environment and humans, which leads to the emergence of new situations that cannot be completely anticipated by the manufacturer of a single device. Basic mechanisms to consider safety-critical behaviour on the workflow level were already discussed in the previous section with respect to *Time-dependant Behaviour*. Currently, we do not support the explicit definition of preconditions and postconditions–possibly related to safety-critical criteria–regarding the execution of individual process steps. The investigation of this aspect may be subject to future work. However, events and EPL statements can be used to define safety-relevant preconditions as process steps to be activated before the safety-critical process step. Goals can be used to define success and error conditions regarding the execution of process steps to be checked during and after the execution within the MAPE-K loops. By specifying these preconditions and postconditions on the workflow level, we are able to flexibly use and compose additional data from external sensors (e. g., light barriers or distance sensors) and other devices (e. g., cameras or robots) to be linked to the execution of particular process steps, and to check the compliance with safety-relevant context factors as preconditions and postconditions. We also discussed more advanced concepts to realize safety-critical behaviour related to the implementation of *Cyber-physical ACID* and *Cyber-physical Transactions* (cf. Section 6.9), possibly applying *Roles* for handling concurrent access to physical resources.

Data security is an important issue regarding the WfMS and also the other software components, especially the IoT middleware collecting sensor and actuator data. When transferring data between devices and components, we assume that the connection is properly secured (e. .g, by using SSL/TLS or SSH). Persistent secure data storage is only partially related to the implementation of a CPS WfMS, which is why it is out of scope of our work. Standard encryption mechanisms can be applied where needed. In case of the distributed execution of a subprocess, subcontracting provides us with a basic means of additional data security as only the data required to execute a specific instance of the subprocess is transferred to the remote peer running the WfMS. More advanced mechanisms for implementing safety and security for CPS workflows in the smart home domain and in other domains have to be investigated in future research. Especially in more safety and time-critical environments (e. g., production, automotive or avionics), the suitability of the WfMS developed in this work needs to be re-evaluated and further investigated as these domains may require additional software components and new concepts.

Advantages:

- Safety related preconditions and postconditions for process activities can be defined in goals or event process steps.
- Checking of safety-related criteria in MAPE-K loops

- Data security through subcontracting for distributed processes

Limitations:

- Safety-critical processes left to be considered closer to the CPS hardware (outside workflow layer)
- Cyber-physical transactions only on conceptual level
- No guarantees/contracts w. r. t. to safety or security
- No explicit focus on safety and security

7.6.11. Summary

Table 7.17 gives a brief summary of the qualitative discussion regarding the fulfilment of the requirements *R1–R8* identified as key requirements for designing a WfMS for CPS in Section 2.6. The abstraction and integration of multiple complex sensors in business processes with the help of CEP leads to a very good coverage and fulfilment of requirement *R1* (Complex Sensors) as special feature of the PROtEUS WfMS. The unified service-based access and dynamic discovery of resources using semantic data within the SAL also contributes to a very good coverage of requirement *R2* (Dynamic Services) as special feature of the PROtEUS WfMS. With the explicit integration of manual tasks into business processes and flexible communication with new process management applications–but no explicit focus on usability–we achieve a good support of requirement *R3* (Human Interaction) within the PROtEUS WfMS. The autonomous execution of process fragments in a scalable hierarchical infrastructure via subcontracting, but relatively simple distribution and management algorithms as well as no considerations of transaction safety, lead to a good fulfilment of requirement *R4* (Distributed Processes) with the D-PROtEUS and PROtEUS WfMSes. A very good coverage of requirement *R5* (CPS Sync) as a special feature is achieved via the correlation of workflows and physical effects with the help of sensors in MAPE-K feedback loops by the Feedback Service and via the flexible specifications of success criteria for *Cyber-physical Consistency* in goals. The error detection based on physical sensors in combination with success and failure criteria specified in goals for the process execution, as well as process adaptations in feedback loops to remedy cyber-physical errors lead to a very good fulfilment of requirement *R6* (CPS Errors) via the Feedback Service. MAPE-K control loops in combination with goals serve as a framework for enabling self-management of the WfMS with respect to arbitrary criteria and self-* mechanisms–achieving a good coverage of requirement *R7* (Self-*). Complex extensions, compensation strategies and goal definitions are necessary to apply the Feedback Service in other contexts, though. With the MAPE-K-based Feedback Service as a general component to extend existing service-based WfMSes with self-management capabilities through minor modifications to the existing processes or WfMSes, we achieve a very good coverage of requirement *R8* (Retrofit) as a special feature of our concepts. Despite being out of focus of our work, time and safety-relevant conditions for the process execution can be specified and checked using *Escalation Ports*, *Events* and *Goals*.

7.7. Comparison with Related Work

The direct analysis of the requirements fulfilment compared to the two most relevant related works–*SmartPM* by Marella et al. [MMS16] and *SitOPT* by Wieland et al. [MMS16]–is shown in Figure 7.38. From this comparison, we see advances of our concepts and prototypes concerning the development of a CPS WfMS in the areas of complex sensor processing, dynamic resource selection, human interactions, the remedy of cyber-physical errors and retrofitting of existing WfMSes.

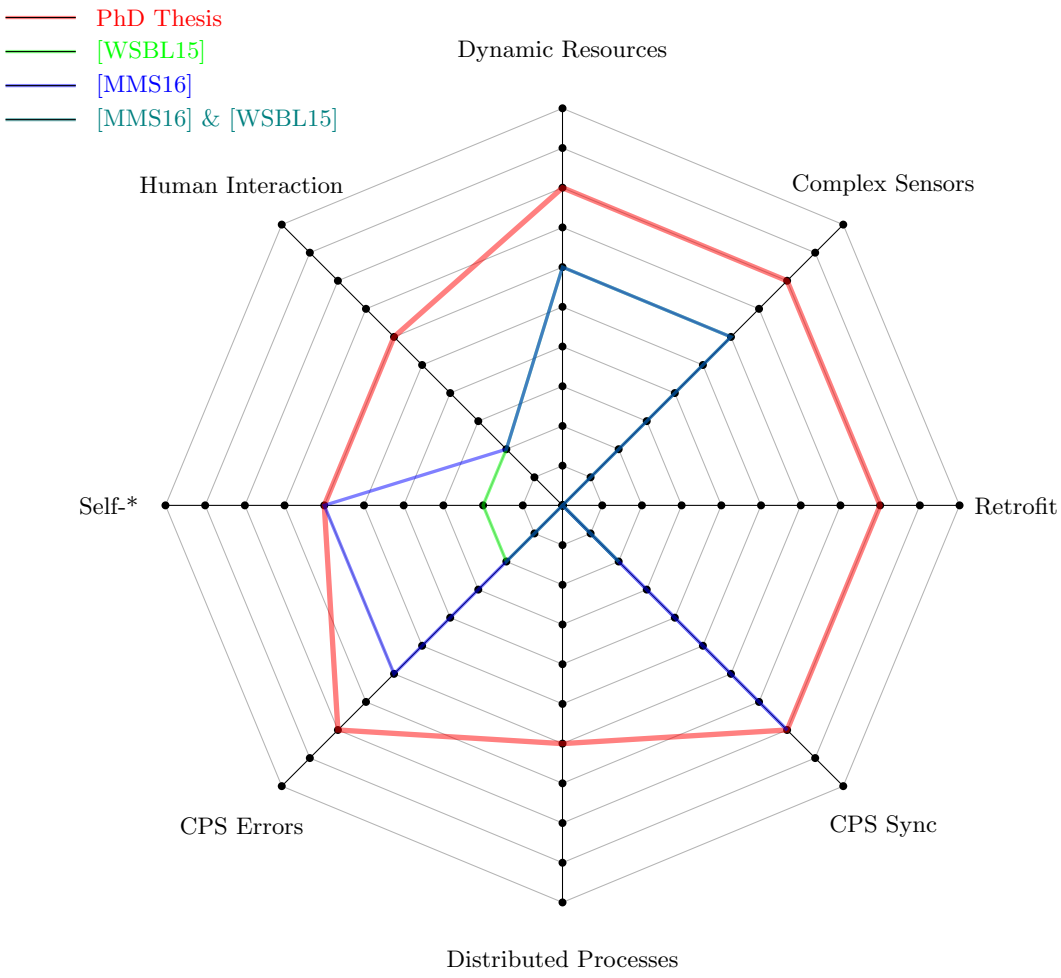


Figure 7.38.: Requirements Coverage of this PhD Thesis Compared to the two most Relevant Related Works [WSBL15] and [MMS16].

Table 7.18 presents a summarizing evaluation of this thesis and highlight of its advances with respect to the requirements and compared to related work. We distinguish between four levels of support regarding the fulfilment of the individual requirement by the respective approach: Special Feature/Unique Selling Proposition (++); supported (+); partially supported (o); not supported (-). The CPS workflow modelling concepts, system architecture for a CPS WfMS, and application of feedback loops for enabling self-management of the CPS workflow execution, lead

to the manifestation of several special features and unique selling propositions of the concepts and prototypes related to the CPS WfMS developed in this thesis.

The approach of using CEP to define specific event patterns in business processes that lead to the activation of following process steps, as well as for the respective processing of lower level events provides a scalable solution for fast and flexible domain-independent event pattern detection in business processes comparable to the proposal of Baumgraß et al. [BBDC+15, BCD+15]. Having an ontology that describes the properties and relations of CPS entities semantically allows us to dynamically select appropriate process resources to execute specific tasks at runtime. Compared to related approaches for dynamic resource selection in the context of CPS and business processes [MMS16, WSBL15], we are more flexible due to the ability to exploit the structure and context of the CPS described by the semantic models and knowledge [RSI+17]. Human interactions and providing modern applications and user interfaces for process management are still neglected topics in the current BPM domain. Besides Yousfi et al. [YBSD16] and Giner et al. [GCFP10] putting an explicit focus on new ubiquitous interaction devices and sensors as well as user context in business processes from the human-computer-interaction perspective, we cover the topic of human interactions to a high degree by providing mobile and stationary process management applications to be used in the sense of ubiquitous BPM. The topic of distributed processes is covered extensively for pervasive environments and IoT in [MMG08] and [MCS16], and in more general in [Fri11]. We propose some basic mechanisms for process distribution with a special focus on leveraging hierarchical network structures for a scalable but managed process execution infrastructure, rather than having a completely decentralized device network.

By defining the effects of the workflow execution on the physical world in goals based on success and failure criteria and a formalized model of the CPS, its entities and contexts, we are able to specify the criteria for cyber-physical synchronization similar to Marella et al. [MMS16]. Other approaches do not support this explicit specification. We provide an extensible way of defining compensation strategies to be used for exception treatment in MAPE-K loops when executing CPS workflows. Compared to *SmartPM* [MMS16] and *SitOPT* [WSBL15], our approach is more suitable for dealing with CPS-related errors, and more flexible and extensible regarding the introduction of new compensation strategies and error criteria regarding the self-management of workflows in general. Other approaches for autonomous workflows (e. g., proposed by Oliveira et al. [OCEP13] or Hoenisch et al. [HSDV13]) do not consider the specific properties of CPS and IoT environments. The aspect of retrofitting existing WfMSes with self-* capabilities in the context of CPS has not been discussed by any other related approaches. Only Parekh et al. discuss a very generic framework for retrofitting autonomic capabilities onto legacy systems [PKGV06]–not related to WfMSes. With our work, we show multiple ways of adding these capabilities based on the MAPE-K framework to existing "legacy" WfMSes. In general, many related approaches propose their own extensions and implementations of workflow languages and systems with respect to CPS-related aspects, which ties their proposals to a specific version of modelling notation and prototype [CSB16]. Although we also present a proprietary workflow metamodel and management system for CPS in this work, we discuss the extension of existing related approaches and systems with respect to applying our concepts on a more

general level. We suggest general extensions, system (reference) architectures and frameworks that can be easily abstracted and applied to other WfMSes. Figure 7.38 shows a direct comparison with respect to the requirements coverage of this thesis compared to the two most closely related works by Marella et al. [MMS16] and Wieland et al. [WSBL15]. All in all, the modelling concepts, system architecture and framework proposed in this thesis cover the identified requirements for introducing workflows to CPS to a higher degree than other related approaches and systems.

7.8. Conclusion

In this chapter, we first presented an extensive case study involving multiple experiments conducted in the context of a smart home. To demonstrate the practical applicability of the developed CPS workflow modelling notation and the corresponding CPS WfMS (*PROtEUS*), we evaluated the two complex scenario processes–*Morning Routine* and *Emergency* process–from Section 2.2. By linking the process step executions with the corresponding effects in the physical world via sensors and actuators, we were able to verify the correct execution of processes and interactions of the individual components of the *PROtEUS* base system in combination with the SAL to fulfil the requirements *R1*–*R4*: Complex Sensors, Dynamic Resources, Human Interaction and Distributed Processes. From the complex scenario processes, we extracted some critical process steps that are likely to fail to due errors and inconsistencies from the interactions with the physical world. Those process steps were further investigated in following experiments, being executed within dedicated processes and supervised by the Feedback Service. With the MAPE-K feedback loops, we have an advanced mechanism of linking the workflow executions to their physical effects via sensor data and also detect and remedy errors and inconsistencies. The experiments have shown that the implementation of the MAPE-K control loop and its interaction with the PROtEUS system work as expected leading to reduced errors rates and therefore, requirements *R5*–*R8* can also be fulfilled: CPS Sync, CPS Errors, Self-* and Retrofit. Based on the real-world experiments in the smart home, we were able to show a working proof-of-concept implementation of our CPS WfMS and associated components. The benchmarks show a reasonable performance for the WfMS to be applied in a smart home context and other related CPS domains. More real-time and safety-critical domains may not be fully supported, though.

The qualitative discussion of our concepts and their comparison with related work have shown a high degree of requirements coverage and various advancements of our approach compared to related concepts. We see a very good fulfilment of requirements *R1* (Complex Sensors), *R2* (Dynamic Resources), *R5* (CPS Sync), *R6* (CPS Errors) and *R8* (Retrofit) as special features and contributions of our concepts. Requirements *R3* (Human Interaction), *R4* (Distributed Processes) and *R7* (Self-*) are also covered to a sufficient degree. With our concepts and implementation, we propose a comprehensive WfMS suitable to be applied in the context of CPS. The discussion of the WfMS and concepts has shown a lot of advantages of our work compared to related approaches. It also helped to identify current limitations, open issues and starting points for future research (e. g., regarding the simplification of workflow modelling, increasing the level of flexibility of the modelled workflows, and supporting real-time and safety-critical behaviour).

Table 7.17.: Summarizing Evaluation of Requirements Fulfilment.

Requirement	Fulfilment	Explanation
R1: *Complex Sensors*	++	Sensor abstraction and integration into business processes using complex event processing
R2: *Dynamic Resources*	++	Unified service-based access and dynamic discovery of resources using semantic data
R3: *Human Interaction*	+	Explicit integration of manual tasks into business processes and flexible communication with new process management applications; no focus on usability
R4: *Distributed Processes*	+	Autonomous execution of process fragments in a scalable hierarchical network structure via subcontracting; simple distribution and management algorithms and no considerations of transaction safety
R5: *CPS Sync*	++	Correlation of workflows and physical effects via sensors in MAPE-K feedback loops and flexible specification of consistency, success and error criteria in goals
R6: *CPS Errors*	++	Error detection based on physical sensors in combination with success and failure criteria for process execution, and process adaptations in feedback loops to remedy CPS errors
R7: *Self-**	+	MAPE-K control loop in combination with goals as framework for enabling self-management of the WfMS with respect to arbitrary criteria and self-* mechanisms; complex extensions, compensation strategies and goal definitions necessary
R8: *Retrofit*	++	MAPE-K-based Feedback Service as general component to extend existing service-based WfMSes with self-management capabilities by adding minor modifications to the existing processes or WfMS

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

Table 7.18.: Related Work and Advances of this Thesis related to Requirements.

Work \ Req.	R1 Complex Sensors	R2 Dynamic Resources	R3 Human Interaction	R4 Distributed Processes	R5 CPS Sync	R6 CPS Errors	R7 Self-*	R8 Retrofit
[DMC14]	+	o	o	-	-	-	-	-
[KSKP11]	+	-	o	-	-	-	-	-
[SSOK13]	+	+	o	o	-	-	-	-
[BBDC+15]	++	o	o	-	-	-	-	-
[AKF+14]	++	o	o	-	-	-	-	-
[MRM13]	+	+	o	-	-	-	-	-
[YBSD16]	+	o	++	-	-	-	-	-
[GKGK16]	o	+	o	-	-	-	-	-
[BDGP17]	+	o	o	-	-	-	-	-
[MD17]	o	o	o	-	-	-	-	-
[SGCG17]	o	+	+	-	-	-	-	-
[GEPF11]	o	o	-	-	-	-	-	-
[CS11]	+	o	-	-	-	-	-	-
[JDK15]	o	-	o	-	-	-	-	-
[MMG08]	-	++	-	++	-	-	-	-
[DTB+15]	+	+	o	+	-	-	+	-
[PRBA15]	+	o	+	+	-	-	-	-
[GCFP10]	+	o	++	o	-	-	-	-
[PRS+13]	+	+	o	+	-	-	-	-
[MCS16]	+	+	o	++	-	-	-	-
[PLM16]	o	-	o	-	o	-	-	-
[Sto15]	-	-	-	-	+	-	-	-
[CR15]	-	+	o	-	+	-	-	-
[RvWLB15]	-	-	-	-	+	-	-	-
[RSI+17]	-	++	+	-	o	-	-	-
[DRSA12]	+	o	-	-	+	+	-	-
[Wom11b]	+	-	-	-	+	o	-	-
[MDCM17]	+	o	o	-	+	-	-	-
[RBD+09]	+	+	-	-	-	-	++	-
[BDK+15]	o	o	-	-	o	o	++	-
[MMP06]	-	+	-	-	o	o	+	-
[Fri11]	-	o	-	++	-	-	+	-
[CL08]	-	+	-	-	-	-	o	-
[HSDV13]	o	+	-	o	-	-	++	-
[OCEP13]	o	o	-	-	-	-	+	-
[RSA10]	o	-	-	-	-	-	++	-
[Sch09]	-	-	+	+	-	-	o	-
[WSBL15]	+	+	o	-	o	o	+	-
[MMS16]	+	+	o	-	++	+	+	-
[XRK08]	o	-	-	-	-	-	-	o
[PKGV06]	+	o	-	-	-	-	+	+
[LBK15]	o	o	-	-	-	-	+	o
[LPS+09]	o	o	-	-	-	-	+	+
PhD Thesis	++	++	+	+	++	++	+	++

++ = Special Feature (USP); + = supported; o = partially supported; - = not supported

8. Summary and Future Work

> "Wisdom comes from experience. Experience is often a result of lack of wisdom."
>
> Terry Pratchett

8.1. Summary and Conclusion

In the beginning of this thesis, we discussed the possible advantages of using BPM technologies to automate and orchestrate processes among systems and devices in the context of CPS and IoT. With the ongoing digitalization and pervasion of almost all areas of everyday live by microcomputers and smart objects, the need for defining and executing flexible and resilient processes in cyber-physical environments (PACPS) on an abstract workflow-oriented level–without hard-wiring and programming the respective applications–rises. Among others, CPS consist of a variety of heterogeneous sensors, actuators, smart objects, more complex devices (e. g., robots or production machines) and also virtual (software) services and applications that interact with humans and the physical environment–and vice versa. Workflow management systems operating in the context of CPS (PACPS) have to support the interaction with these heterogeneous and possibly resource constraint entities in a proactive and reactive manner while being able to detect and react to unanticipated situations and errors in the physical and virtual worlds. Based on the specific properties of CPS and IoT environments (cf. Section 2.4.6) as well as two scenario processes from the smart home domain (cf. Section 2.2), we derived a set of eight requirements a WfMS has to fulfil in order to be used for process execution in CPS (cf. Section 2.6). These requirements address basic functionality of the WfMS to interact with complex sensors (*R1*), dynamic services and resources (*R2*), humans (*R3*) and other WfMSes in a distributed process execution setting (*R4*). These requirements also comprise more advanced aspects to achieve a certain level of self-management enabling cyber-physical resilience, namely the capabilities of cyber-physical synchronization (*R5*) and handling of cyber-physical errors (*R6*), which leads to the more general capability of the WfMS supporting self-management (*R7*) and also adding this capabilities to existing WfMSes in a *Retrofitting Process* (*R8*). Related to these requirements, we derived five central research questions to be investigated in the course of this thesis regarding the modelling of workflows in CPS (*Q1*), the design of a CPS workflow management system (*Q2*), the synchronization of virtual and physical world processes (*Q3*), the adding of self-* capabilities to a CPS WfMS (*Q4*) and the retrofitting of existing WfMSes with self-management capabilities (*Q5*) (cf. Section 2.6.3).

Using the identified requirements and derived research questions as a basis, we conducted an extensive literature study and evaluation that comprises related approaches and WfMSes with respect to: already existing BPM systems in industry and academia; the modelling of CPS workflows; CPS workflow systems; cyber-physical synchronization; self-* for BPM systems; and retrofitting frameworks for WfMSes. We found that most WfMSes already in productive use in industry and academia only fulfil a very small subset of the identified basic requirements to be suitable WfMSes used in the context of CPS. Lots of approaches from related research cover specific aspects and partial subsets of the requirements, also with respect to the self-management and cyber-physical resilience. However, none of the investigated works/WfMSes cover all of the identified requirements sufficiently to be suitable systems for managing workflows in IoT and CPS.

From the findings of investigating related approaches, we started designing and developing a WfMS for CPS in accordance with the BPM lifecycle [VDA13] to address the specific research questions and requirements. Related to the *Design* phase, we developed the basic CPS workflow modelling language to be able to specify concrete processes on a technical implementation-oriented level. The component-based process notation enables the composition of hierarchical processes, subprocesses and atomic process steps, which are connected with each other via typed ports and transitions. Special process steps are used to define the activation of complex events with the help of event patterns, service invocations, human interactions, and dynamic service selection and invocation using semantic knowledge on the business process level. As an extension to these basic elements, the workflow language also supports the specification of the execution outcome in the form of goals and objectives containing success and error criteria with respect to cyber-physical actions and more general arbitrary aspects. Based on this specification, we derived the concept of *Cyber-physical Consistency* to indicate if the assumed/expected state of the process execution corresponds to the actual state. The goal definition for a complete cyber-physical process can be outsourced into a *Consistency Style Sheet* to separate the additional attributes regarding goals and objectives of individual process steps from the "basic" process model. Following, we designed a corresponding system architecture to execute the modelled processes (*Implement/Configure* phase): the *PROtEUS* WfMS. The architecture consists of components for CEP of sensor streams; local and remote service invocations to activate physical and virtual actuators; human interactions and process management; dynamic service selection including a semantic knowledge base (*Semantic Access Layer*); and the distributed execution of subprocesses in a hierarchical network of peers running the basic PROtEUS configuration and managing super-peers running the *D-PROtEUS* configuration. With respect to the *Run & Adjust* phase, we added an additional external software component (*Feedback Service*) that can be called from the basic WfMS. This component uses the goals specified for a specific process step to execute MAPE-K feedback loops for the particular process step to monitor and analyse relevant external sensor data with respect to the success and error criteria defined in the objectives, and to confirm the successful execution or to adapt the process execution in case of errors and undesired situations trying to remedy the occurred issues. *Consistency Levels* can be defined as part of an objective to specify the level of precision and consistency to be achieved during execution of the compensation actions in case of errors, which

may reduce the overall execution time (*Scalable Consistency*). Due to the Feedback Service implementing the MAPE-K feedback loop having been developed as a standalone generic software component, it also allows for extending other existing WfMSes and software systems with self-management capabilities based on the MAPE-K framework regarding arbitrary factors and extensible adaptation strategies to compensate undesired and unanticipated situations. In the context of CPS, we put a special focus on the process resources as the main sources of new errors and adapt the process execution accordingly.

In order to evaluate our concepts and implementations as a proof-of-concept, we conducted several experiments focussing on the WfMS and associated components interacting with sensors, actuators, robots, humans and virtual services in a smart home case study. By linking the effects of the CPS workflow execution to its effects in the physical and virtual worlds via additional sensors, we are able to verify the correct behaviour of the WfMS and its components. The PROtEUS system in its basic configuration as well as the D-PROtEUS system show expected behaviour when executing the scenario processes. The Feedback Service applying the MAPE-K control loop to specific process steps shows a decrease of errors and imprecisions for error prone processes–demonstrated for cyber-physical and distributed processes involving robots, lights and a coffee maker. The times measured for the execution of individual process instances attest the WfMS a suitable performance for executing processes in the context of a smart home. The durations of virtual computations are usually in the order of a few milliseconds. Performing a physical process activity takes much longer time due to physical processes by nature being much slower than a computer. The results of the following qualitative discussion have shown a high degree of fulfilment for the addressed requirements by our concepts, with some special unique features (e. g., with respect to the dynamic service discovery, cyber-physical synchronization or retrofitting) compared to related approaches. We were also able to identify limitations of our concepts, which may serve as starting points for future work, e. g., related to the simplification and increase of flexibility for goals, EPL statements and SPARQL queries; the application of more sophisticated mechanisms for process distribution and adaptation; the realization and ensuring of time and safety-critical behaviour; and the conduction of large scale experiments involving multiple concurrent process instances in other CPS domains. All in all, the concepts and implementations related to the development of a WfMS for CPS presented in this thesis prove to be feasible for introducing the concept of workflows into CPS and IoT environments, which allows us to leverage the advantages of having a flexible and easy way of specifying and configuring resilient automated processes on top of cyber-physical devices, objects, humans, applications, services and systems.

In summary, we were able to address all requirements identified for introducing BPM technologies into CPS. CEP and dynamic service selection based on a semantic model of the CPS entities allow for dealing with complex streams of sensor data (Requirement *R1*) and dynamic availability of resources (Requirement *R2*) on the business process level. Human interactions (Requirement *R3*) are addressed mostly on the technical level by providing new and innovative means and applications for the management of and interaction with and as part of the business processes. Regarding the distributed process execution and interactions of multiple WfMSes (Requirement *R4*) we propose to establish a hierarchical structure of managing super-peers

and regular process execution peers that interchange and execute partial processes. Goals and objectives are used as additional process-related information to specify the effects and consistency criteria for successfully executing a process instance in the physical world (Requirement *R5*). The verification of the process execution outcome is realized based on the MAPE-K loop in combination with external sensors; compensations for determined cyber-physical inconsistencies or errors are derived and executed automatically as part of the MAPE-K executions (Requirement *R6*). Besides realizing this self-healing mechanism for cyber-physical errors based on predefined goals, the MAPE-K control loop can be used as a more general framework to add arbitrary self-x capabilities to workflows and WfMSes (Requirement *R7*). With the implementation of the MAPE-K framework as a standalone service-based software component, we are also able to add these capabilities to existing "legacy" WfMSes by either extending the internal process execution logics and metamodel of the WfMS in an invasive way, or by extending the existing process models with additional service invocations in a non-invasive manner (Requirement *R8*).

8.2. Advances of this Thesis

From the comparison with and evaluation of related work with respect to the requirements *R1–R8* in Section 7.7, we see contributions and advances of our concepts and prototypes with respect to all requirements and in multiple areas regarding the following aspects:

With our extensible process modelling notation, we are able to specify the interaction with heterogeneous and dynamic entities that form a CPS–namely multiple complex sensors, actuators of varying complexity, humans, smart objects, and virtual software services–on the abstract level of business processes. None of the investigated related approaches provides a modelling notation with a comparable degree of expressiveness for CPS. In addition, we are able to specify success and error criteria for the execution of process steps as separate aspects, which allows us to verify and restore cyber-physical consistency at runtime. Our workflow notation is therefore able to fulfil the identified requirements on the modelling level and presents an answer to research question *Q1* regarding the modelling of CPS workflows.

With our component-based (reference) architecture of a CPS WfMS (*PROtEUS*), we are able to implement and execute the modelled CPS workflows interacting with the aforementioned CPS entities. Compared to other approaches, our concepts hereby support a wider range of features (e. g., with respect to human interactions, distributed process execution or dynamic service selection) to fulfil the requirements *R1–R4* on the implementation level to a higher degree, which answers research question *Q2* w. r. t. the design of a CPS WfMS for executing CPS workflows.

The Feedback Service as a standalone software component implementing a generic variant of the MAPE-K feedback loop enables us to add the capability of self-management to the basic WfMS. With that, we are able to check cyber-physical consistency for the execution of individual process steps based on the specified goals as well as to detect and repair errors by adapting the process execution. This component fulfils requirements *R5–R7* in a more flexible, extensible and generic way than other approaches. The MAPE-K concept in combination with goal specifications proved to be a suitable framework for realizing the aspect of cyber-physical

synchronization and adding self-* capabilities to the WfMS, which answers research questions *Q3* and *Q4*. As the Feedback Service is implemented in the form of a standalone micro-service, we can also add the cyber-physical synchronization and self-* capabilities to other existing WfMSes by only requiring minor modifications to the "legacy" process or "legacy" WfMS and its metamodel. This answers research question *Q5*, which is an important advance of our work, as there is a very diverse and heterogeneous landscape of existing WfMSes already in use in industry and academia. Despite it being an important requirement for future WfMSes to be used in the context of CPS and IoT, no other related research addresses the issue of adding autonomous capabilities to existing WfMSes explicitly. With our modelling language, system architecture and autonomous manager it is possible to establish links between the execution of process instances and the cyber-physical environment–and vice versa–to a new extend compared to other approaches.

With answering the research questions and fulfilling all identified requirements by our concepts and prototypes, we are able to verify the automation hypothesis identified in Section 2.6.3 and provide a holistic solution for using workflows in CPS to facilitate the linking of components, increase the level of automation, and enable resilient autonomous processes. Compared to related approaches, we developed a comprehensive WfMS for CPS enabling a higher degree of automation and autonomy in CPS via resilient processes on top of the individual CPS components. Due to the successful implementation of workflows for CPS and IoT, we are able to leverage the advantages of using workflow technologies discussed in Section 2.5.2 in the context of CPS, e. g., the flexible combination of active and reactive functionality across devices and systems; the easy integration of heterogeneous systems on the workflow layer without alterations to these systems; the high-level programming of workflows oriented towards end-user programming; or the abilities to document, analyse and optimize processes. With our CPS workflow notation, CPS WfMS and implementation of the MAPE-K feedback loop, we are able to address all identified requirements and research challenges for applying BPM technologies in CPS in the scope of this thesis. We focussed on the integration aspects regarding the combination of existing and newly developed software components as part of the software and systems engineering processes. Special attention was paid to the interactions of the WfMS with the physical world resulting in new concepts regarding *Cyber-physical Synchronization* and *Cyber-physical Consistency*.

The increased level of automation is listed as one of the advantages that the application of BPM technologies in CPS entails. The main purpose of (business) processes is to enable the automation of manual tasks to increase efficiency and decrease the need for manual work. While in some CPS domains this is the desired outcome of the CPS workflow applications (e. g., in the smart factory), other domains (e. g., the smart home) are centred around humans and their activities, which can only be partially automated and supported via workflows. The users have to be in control of the CPS all the time. They have to be provided with information about process instances currently being executed and they have to be able to control all processes as well as the CPS devices. However, workflows and the associated automations of activities can also provide advantages in the smart home/AAL domain as shown with the *Emergency* scenario processes (cf. Section 2.2.2). Here we assume that the resident is not able to perform manual tasks anymore due to health issues and

the WfMS supports the user by calling the emergency service and opening the door automatically according to the underlying process. Hence the CPS workflows have to be carefully identified and designed by the domain expert to add value to having automated workflows supporting users with everyday tasks and routines (e. g., in case of their absence or emergencies). In the smart home, the modelled workflows provide the basic recipes of the automated activities; to cope with the need for flexibility and adaptations at runtime (cf. Section 2.4.6) the MAPE-K feedback loop for autonomous systems is applied to the execution. In the domain of smart factories a high-level of automation without human interventions and highly adaptive and flexible CPS entities (production machines, robots, etc.) are the desired goals [Jaz14]. CPS workflows in combination with the MAPE-K loops as proposed in this thesis can also help to achieve this high level of automation and flexibility. However, humans will still play an important role in Industry 4.0 in the roles of supervisors and co-workers [GLR+15], which requires context-adaptive and safety-aware workflows. In summary, we can observe a conflict between the need for dependable processes according to the requirements of the individual domain on the one hand; and on the other hand there is a need for flexibility to cope with the unpredictable and not fully controllable nature of CPS. We try to address these contradictory requirements with our concepts and prototypes for a CPS WfMS and the associated MAPE-K-based autonomic manager.

8.3. Contributions to the Research Area

As already mentioned in the introductory sections of this thesis, the application of BPM technologies in the context of IoT and CPS is still a young and vibrant research field. With this work, we present a comprehensive approach of introducing workflow technologies to CPS and IoT with the aim of increasing the level of automation in these environments. Besides an extensive discussion of a first set of requirements related to the application of workflows in CPS and the corresponding evaluation of related work to identify potential research gaps, our main contributions to the research area of BPM in CPS comprise:

- A domain-independent modelling notation for executable workflows and entities in CPS that supports the specification of the process outcome using Consistency Style Sheets;
- A system architecture of a distributed WfMS for CPS able to execute CPS workflows interacting with the physical world via sensors, actuators, smart objects and humans (*PROtEUS* and *D-PROtEUS*);
- A generic software component for adding multi-level feedback loops founded on the MAPE-K concept to workflows and WfMSes to enable cyber-physical synchronization based on the concept of *Cyber-physical Consistency* and self-management with scalable precision (*Feedback Service*);
- A retrofitting framework and processes for extending existing WfMSes with (autonomic) self-* capabilities in an invasive and non-invasive way by using MAPE-K feedback loops.

The overall objective of this thesis is the development of a holistic WfMS that can be used in the context of CPS and IoT. We developed concepts and prototypes to model, implement and enact complex interactions among all the involved CPS entities–sensors and sensor compounds, actuators and actuator compounds, smart objects, humans, software services and applications–and with the physical environment on a business process-oriented level in PACPS. The CPS workflows and CPS WfMS are able to react to dynamic changes in the structure of the CPS and to increase fault-tolerance and resilience due to their autonomic capabilities. With our modelling language, system architecture and autonomous manager it is possible to establish links between the execution of process instances and the cyber-physical environment, and vice versa. Our focus is on addressing the identified requirements *R1–R8* (cf. Section 2.6) in sum–not isolated–with a holistic WfMS that integrates well-established components and technologies together with self-developed components and engineering concepts into a WfMS for CPS and IoT. The individual software components of the CPS WfMS and associated services use and implement state-of-the-art technologies (cf. Sections 5.2 and 7.2). With our concepts, we present a new holistic engineering approach and framework for modelling, implementing and executing self-managed workflows in CPS and IoT.

As discussed in Section 7.7, we show with our new concepts regarding the development of a CPS WfMS advancements with respect to various aspects across all phases of the BPM lifecycle with a special focus on CPS-related properties and concepts. The quantitative evaluation presents a possible way of evaluating the feasibility of the proposed concepts and their implementations within a case study in the context of a smart home as a CPS example. From the qualitative discussion, we are able to identify advantages and also limitations and open issues regarding our new concepts, which can be used as relevant starting points and basis for future developments in the field of BPM for CPS and IoT.

8.4. Relevance

In the beginning of this thesis, we discussed the importance of being able to specify and execute flexible processes in the context of Industry 4.0 involving future CPS and IoT environments. With our work, we present various concepts to implement workflows in cyber-physical domains to achieve the flexibility and resilience required by emerging smart spaces that consist of software and physical world components interacting with each other. With our work, we present new concepts regarding specific aspects with respect to the modelling and execution of workflows in CPS, but also a holistic approach to implement workflows in CPS along the BPM lifecycle. The comparison of our work with related approaches shows advances of our concepts regarding various aspects and an in-depth discussion of the topic of CPS workflows to a new extend. This work is highly relevant for both the fields of BPM and CPS/IoT research, but also shows more general contributions to the fields of software engineering, systems engineering and architecture, and human-computer-interaction. We were able to prove this relevance with the help of multiple peer-reviewed publications at respective renown scientific conferences.

In Section 2.5.3 we briefly mention relevant challenges showing the interaction between IoT and BPM identified by Janiesch et al. of the BPM community in [JKM+17]. With our work, we see contributions regarding the following challenges:

- C1: Placing sensors in a process-aware way (by retrofitting machines/devices with sensors and using them in processes in MAPE-K feedback loops)
- C2: Monitoring of manual activities (by using additional environmental sensors and interactive devices)
- C3: Connection of analytical processes with IoT (by analysing the CPS process execution in feedback loops and with event patterns)
- C4: Integrating IoT into the correctness check of processes (by using additional IoT sensors in feedback loops to verify process executions)
- C5: Dealing with unstructured environments (by using feedback loops to adapt processes at runtime in case of new and undesired situations)
- C9: Specifying the autonomy level of things (by defining the respective goals for the process execution to be used in feedback loops)
- C11: Concretizing abstract process models (by using our more technical workflow notation for CPS)
- C12: Dealing with new situations (by applying feedback loops to adapt processes at runtime in case of new situations)
- C13: Bridging the gap between event-based and process-based systems (by applying CEP-based event pattern detection at the process level)
- C14: Improving online conformance checking (by applying feedback loops with additional data to verify the process execution)
- C15: Improving resource utilization optimization (by distributing process fragments to special peers and defining optimization criteria in goals)
- C16: Improving resource monitoring and quality of task execution (by using the respective data in feedback loops to analyse and adapt the processes).

In addition to the previous discussion of challenges for BPM and IoT from the point of view of the BPM community with respect to our work, Figure 8.1 shows a research roadmap for *Business Process Management Systems for Internet of Things (BPMS4IoT)* drawn up by Chang et al. from the systems architecture community [CSB16]. With our work, we see contributions regarding the following aspects:

- Generic IoT-driven BP model specification (by using our more technical workflow notation for CPS)
- Edge computing-driven BP modelling (by defining specific resources to execute certain parts of a distributed process)
- Hybrid (Fog/Mist) BPMS architecture (by using a hierarchical overlay network of peers and super-peers for distributed process execution)
- Resource-awareness (by having a semantic model describing the properties of and relations among process resources)

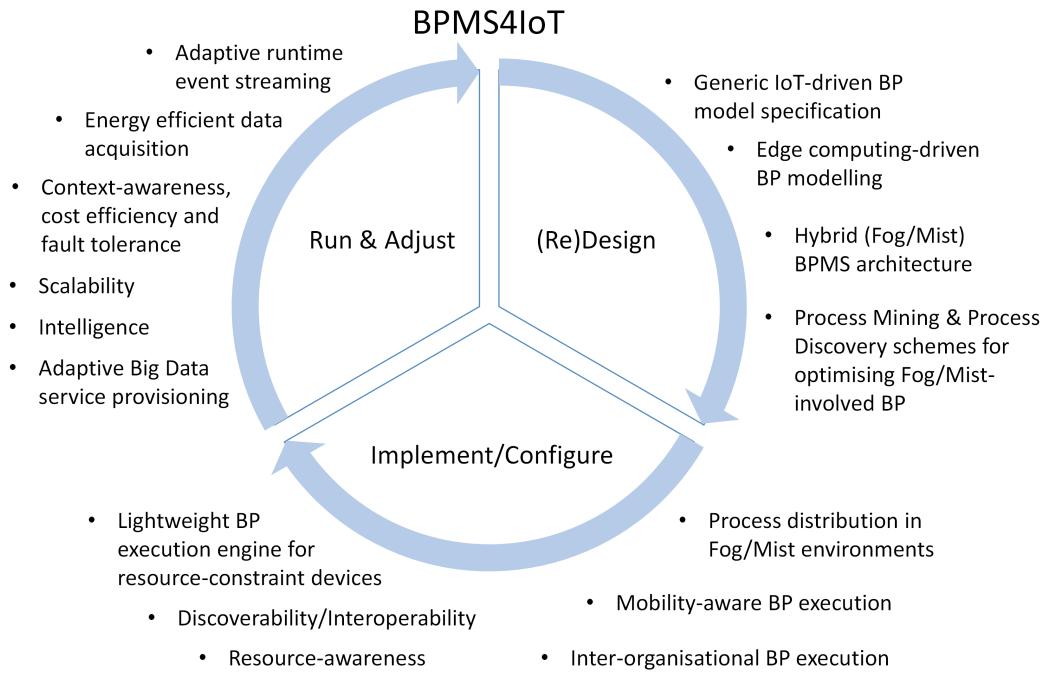


Figure 8.1.: Research Roadmap in BPMS for IoT from [CSB16].

- Discoverability/interoperability (by using semantic data to dynamically discover resources/services and having a service-based middleware to provide unified access)
- Process distribution in Fog/Mist environments (by utilizing subcontracting for distributed process execution in a hierarchical network)
- Inter-organisational BP execution (by utilizing subcontracting to exchange process fragments among the involved resources in distributed processes)
- Mobility-aware BP execution (by defining specific mobility/location-related process execution aspects in goals and using them in MAPE-K feedback loops)
- Scalability (in various ways: by supporting distributed process execution; by defining consistency levels; by using Consistency Style Sheets)
- Context-awareness, cost efficiency and fault tolerance (by using the related data in feedback loops and adapting the respective processes).

8.5. Open Questions

A detailed discussion regarding the limitations of the individual concepts and prototypes with respect to the respective requirements can be found in Section 7.6.

Open questions concerning the modelling (design) of CPS workflows comprise:

- the evaluation of the CPS workflow notation with respect to the workflow patterns [vDATHKB03] and their extension towards *Workflow Patterns for CPS*;
- the use of more declarative modelling concepts [RSS13];
- the application of the CPS modelling language to other CPS domains;
- the application of the proposed modelling concepts to existing workflow notations (e. g., WS-BPEL or BPMN 2.0);
- the simplification of the modelling tools and concepts towards end-user modelling of CPS workflows;
- the automated learning (synthesis) and dynamic adaptation of specific workflow attributes (e. g., EPL patterns or goals) and entire CPS workflows.

Open questions concerning the implementation of the CPS WfMS comprise:

- the formal verification of the execution behaviour of the process engine based on Petri nets;
- the evaluation of the proposed WfMS in large scale experiments, also with a more complex hierarchical peer–super-peer network;
- the application of the system architecture as a reference architecture for other CPS domains;
- the investigation of multiple workflows and instances being executed in parallel in the same CPS and with that, the handling of concurrency, implementation of cyber-physical transactions, and the optimization of task scheduling.

Open questions related to the running and adjustment of CPS workflows comprise:

- the integration of more sophisticated algorithms regarding the analysis of the CPS process execution (possibly in combination with online process mining and conformance checking);
- the development of an extensive classification of cyber-physical errors and corresponding rollback actions/compensations/remedies;
- the application of more sophisticated algorithms to derive strategies for compensation of errors and undesired behaviour in the planning phase of the MAPE-K loop at runtime (e. g., based on logics or planning [Mar17]);
- the integration of more complex process adaptation strategies, including the evolutionary learning of instance adaptations and process model adaptations (e. g., based on machine learning);
- the deeper investigation and implementation of concepts regarding security, safety and real-time.

With respect to the challenges concerning the interaction between IoT and BPM discussed by Janiesch et al. in [JKM+17], we mostly see open questions regarding the detection of new processes from data (*C8*), breaking down end-to-end processes (*C7*) and managing the link between micro-processes (*C6*), because we currently rely on

the workflow designers to specify most of the processes and process landscape. We also did not investigate the "social" role of agents in the phase of prediction and adaptation (*C10*). Other challenges were only discussed briefly or we provide only little contributions respectively, e. g., regarding the autonomy level of things (*C9*), online conformance checking (*C14*) or improving resource utilization and quality of task execution (*C15, C16*).

Regarding the research roadmap for *BPMS4IoT* by Chang et al. [CSB16], we see open research topics related to the process mining and discovery schemes for Fog/Mist involved business processes and the implementation of a lightweight business process execution engine. Other subjects that we not addressed explicitly include the adaptive big data provisioning, intelligence, energy efficient data acquisition, and adaptive runtime event streaming. These topics are not only relevant in the context of BPMS, but for IoT and CPS in general.

8.6. Future Work

From the open questions discussed in the previous section, we see a lot of potential for future developments and extensions of our concepts and prototypes regarding the application of BPM technologies in the context of CPS and IoT. Our next steps will mostly focus on the improvement of the augmented reality app *HoloFlows* presented in Sections 4.8.2 and 5.6.3 to simplify the modelling of more complex CPS workflows and configurations for end-users. We will also investigate the use of more flexible and declarative workflow modelling concepts as well as the application of more sophisticated adaptation and analysis mechanisms to adjust the workflows at runtime while still conforming to general constraints defined with the workflow model and by the CPS themselves. This includes more dynamic EPL, SPARQL and goal definitions, and also applying the MAPE-K loop on the meta-levels to adapt the respective parameters at runtime. The automated derivation and synthesis of CPS workflows from recorded activity logs and event streams (*Cyber-physical Process Mining*) is also a highly interesting topic for future research. The discussion and implementation of advanced compensation strategies for dealing with unforeseen errors from the physical world in CPS and in more general the realization of cyber-physical transactions are necessary steps to mature the presented concepts and establish BPM technologies in CPS and IoT in the future. A list of additional open issues regarding research in BPM and CPS/IoT not covered in this thesis can be found in Section 8.5.

The next steps also include the application and deployment of our prototypes in other domains, specifically in the context of smart facility management and smart manufacturing to model and execute respective CPS workflows in Industry 4.0 scenarios. With the help of these larger scale case studies, we will be able to evaluate the requirements fulfilment and performance of the prototypes and also to identify new issues and requirements that have to be met in these new cyber-physical domains (e. g., related to safety and security). Along with these larger scale studies, we will put a special focus on the execution of multiple process instances and feedback loops in parallel, which will possibly influence each other. Here the correlation of events occurring in CPS with the execution of the corresponding process instances is a difficult challenge that has to be addressed. The application and adaptation of

the *OPC-UA* standard [LM06] may simplify the implementation of CPS workflows in industrial contexts and already provide built-in mechanisms to realize parts of our proposed concepts. We will also work on the application and tighter integration of our concepts and components to existing open source WfMSes–primarily to Activiti, the YAWL engine, Apache ODE and IBM Node-RED, and thereby further advance the establishment of BPM technologies for CPS and IoT.

Bibliography

[AAD+07] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Gerhard Pfau, et al. Web services human task (ws-humantask). *White Paper*, 2007.

[AAH98] Nabil R Adam, Vijayalakshmi Atluri, and Wei-Kuang Huang. Modeling and analysis of workflows using petri nets. *Journal of Intelligent Information Systems*, 10(2):131–158, 1998.

[ABD+16] Rajeev Alur, Emery Berger, Ann W Drobnis, Limor Fix, Kevin Fu, Gregory D Hager, Daniel Lopresti, Klara Nahrstedt, Elizabeth Mynatt, Shwetak Patel, et al. Systems computing challenges in the internet of things. *arXiv preprint arXiv:1604.02980*, 2016.

[AE14] Bilgin Avenoğlu and P Erhan Eren. A context-aware and workflow-based framework for pervasive environments. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–23, 2014.

[AFG+07] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. Context-aware workflow management. In *Intern. Conference on Web Engineering*, pages 47–52. Springer, 2007.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[AKF+14] Stefan Appel, Pascal Kleber, Sebastian Frischbier, Tobias Freudenreich, and Alejandro Buchmann. Modeling and execution of event stream processing in business processes. *Information Systems*, 46:140–156, 2014.

[Ald03] Frances K Aldrich. Smart homes: past, present and future. In *Inside the smart home*, pages 17–39. Springer, 2003.

[All03] OSGi Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[AMkP04] Chadia Abras, Diane Maloney-krichmar, and Jenny Preece. User-centered design. In *In Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*. Publications, 2004.

[AMT+12] Anil Aswani, Neal Master, Jay Taneja, David Culler, and Claire Tomlin. Reducing transient and steady state electricity consumption in hvac using learning-based model-predictive control. *Proceedings of the IEEE*, 100(1):240–253, 2012.

[ATHEVDA06] Michael Adams, Arthur HM Ter Hofstede, David Edmond, and Wil MP Van Der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 291–308. Springer, 2006.

[ATHVDAE07] Michael Adams, Arthur HM Ter Hofstede, Wil MP Van Der Aalst, and David Edmond. Dynamic, extensible and context-aware exception handling for workflows. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 95–112. Springer, 2007.

[BAJ17] Borja Bordel, Ramón Alcarria, and Antonio Jara. Process execution in humanized cyber-physical systems: Soft processes. In *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on*, pages 1–7. IEEE, 2017.

[BBDC+15] Anne Baumgraß, Mirela Botezatu, Claudio Di Ciccio, Remco Dijkman, Paul Grefen, Marcin Hewelt, Jan Mendling, Andreas Meyer, Shaya Pourmirza, and Hagen Völzer. Towards a methodology for the engineering of event-driven process applications. In *Int. Conf. on Business Process Management*, pages 501–514. Springer, 2015.

[BBDL+13] Martin Bauer, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Andreas Nettsträter, Julinda Stefa, and Joachim W Walewski. Iot reference model. In *Enabling Things to Talk*, pages 113–162. Springer, 2013.

[BC08] Dario Bonino and Fulvio Corno. Dogont-ontology modeling for intelligent domotic environments. In *International Semantic Web Conference*, pages 790–803. Springer, 2008.

[BCD+15] Anne Baumgraß, Claudio Di Ciccio, Remco Dijkman, Marcin Hewelt, Jan Mendling, Andreas Meyer, Shaya Pourmirza, Mathias Weske, and Tsun Yin Wong. GET Controller and UNICORN : Event-driven Process Execution and Monitoring in Logistics. *BPM (Demos)*, (i), 2015.

[BCG12] Manfred Broy, MaraVictoria Cengarle, and Eva Geisberger. Cyber-physical systems: Imminent challenges. In *Large-Scale Complex IT Systems. Development, Operation and Management*, volume 7539, pages 1–28. 2012.

[BD11] Paolo Bocciarelli and Andrea D'Ambrogio. A bpmn extension for modeling non functional properties of business processes. In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 160–168. Society for Computer Simulation International, 2011.

[BDGP17] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio, and Emiliano Paglia. A bpmn extension for modeling cyber-physical-production-systems in the context of industry 4.0. In *IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, pages 599–604. IEEE, 2017.

[BDH+11] Angineh Barkhordarian, Frederik Demuth, Kristof Hamann, Minh Hoang, Sonja Weichler, and Sonja Zaplata. Migratability of bpmn 2.0 process instances. In *International Conference on Service-Oriented Computing*, pages 66–75. Springer, 2011.

[BDK+15] Victor Braberman, Nicolas D'Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. Morph: A reference architecture for configuration and behaviour self-adaptation. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, pages 9–16. ACM, 2015.

[BDK+17] Victor Braberman, Nicolas DIppolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. An extended description of morph: A reference architecture for configuration and behaviour self-adaptation. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 377–408. Springer, 2017.

[BG11] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12:161–166, 2011.

[BGT17] Björn Butzin, Frank Golatowski, and Dirk Timmermann. A survey on information modeling and ontologies in building automation. In *Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE*, pages 8615–8621. IEEE, 2017.

[BI98] Richard R Brooks and Sundararaja S Iyengar. *Multi-sensor fusion: fundamentals and applications with software*. Prentice-Hall, Inc., 1998.

[BK17] A Bhuvaneswari and GR Karpagam. Semantic web service discovery for mobile web services. *International Journal of Business Intelligence and Data Mining*, 13(1-3):95–107, 2017.

[BLLJ98] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. Cascading style sheets, level 2, css2 specification. w3c recommendation 12-may-1998. *World Wide Web Consortium*, 1998.

[BMP13] Antonio Bucchiarone, C Mezzina, and Marco Pistore. Captlang: a language for context-aware and adaptable business processes. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, page 12. ACM, 2013.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[Bör12] Egon Börger. Approaches to modeling business processes: a critical analysis of bpmn, workflow patterns and yawl. *Software and Systems Modeling*, 11(3):305–318, 2012.

[Bor14] Eleonora Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[BPR07] Jörg Becker, Daniel Pfeiffer, and Michael Räckers. Domain specific process modelling in public administrations–the picture-approach. *Electronic Government*, pages 68–79, 2007.

[BR16] Stefan Boschert and Roland Rosen. Digital twin –the simulation aspect. In *Mechatronic Futures*, pages 59–74. Springer, 2016.

[Bro13] Manfred Broy. Engineering cyber-physical systems: Challenges and foundations. In *Complex Systems Design & Management*, pages 1–13. Springer, 2013.

[BS15] Hongyu Pei Breivold and Kristian Sandström. Internet of things for industrial automation–challenges and technical solutions. In *IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, pages 532–539. IEEE, 2015.

[BSMD11] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *Intern. Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.

[BtHS01] Alistair P Barros, Arthur HM ter Hofstede, and C Szyperski. Retrofitting workflows for b2b component assembly. In *25th Annual International Computer Software and Applications Conference, COMPSAC 2001*, pages 123–128. IEEE, 2001.

[Bur14] Sebastian Burckhardt. Principles of eventual consistency. *Foundations and Trends® in Programming Languages*, 1(1-2):1–150, 2014.

[CBF+16] Andrea Ceccarelli, Andrea Bondavalli, Bernhard Froemel, Oliver Hoeftberger, and Hermann Kopetz. *Basic Concepts on Systems of Systems*, pages 1–39. Springer Intern. Publishing, Cham, 2016.

[CDB+12] Marco Conti, Sajal K. Das, Chatschik Bisdikian, Mohan Kumar, Lionel M. Ni, Andrea Passarella, George Roussos, Gerhard Trster, Gene Tsudik, and Franco Zambonelli. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyberphysical convergence. *Pervasive and Mobile Computing*, 8(1):2 – 21, 2012.

[CDK+02] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2):86–93, 2002.

[Cha15] Stuart Chandler. Unlocking the power of the internet of things through bpm. *BPM Everywhere: Internet of Things, Process of Everything*, pages 183–189, 2015.

[Che10] Guihai Chen. Internet of things towards ubiquitous and mobile computing. *Microsoft Research Asia Faculty Summit, Shanghai*, 2010.

[CK11] Alexandru Caracaş and Thorsten Kramp. On the expressiveness of bpmn for modeling wireless sensor networks applications. *Business Process Model and Notation*, pages 16–30, 2011.

[CL08] Chii Chang and Sea Ling. Towards a context-aware solution for device failures in service-oriented workflow. In *Proceedings of the 10th International Conference on Information Integration and Web-Based Applications & Services*, pages 77–83. ACM, 2008.

[CLD+15] Chii Chang, Seng W Loke, Hai Dong, Flora Salim, Satish N Srirama, Mohan Liyanage, and Sea Ling. An energy-efficient inter-organizational wireless sensor data collection framework. In *IEEE International Conference on Web Services (ICWS)*, pages 639–646. IEEE, 2015.

[Coa96] Workflow Manage Coalition. Terminology & glossary. *WFMC Document WFMCTC-1011, Workflow Management Coalition, Avenue Marcel Thiry*, 204:1200, 1996.

[Cor16] Angelo Corsaro. Cloudy, foggy and misty internet of things. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 261–261. ACM, 2016.

[CPFJ04] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *The First Annual Intern. Conf. on Mobile and Ubiquitous Systems: Networking and Services, MOBIQUITOUS*, pages 258–267. IEEE, 2004.

[CR15] Angelo Croatti and Alessandro Ricci. Programming abstractions for augmented worlds, 2015.

[CS11] Wei-Chih Chen and Chi-Sheng Shih. Erwf: Embedded real-time workflow engine for user-centric cyber-physical systems. In *IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 713–720. IEEE, 2011.

[CSB15] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Mobile cloud business process management system for the internet of things: Review, challenges and blueprint. *arXiv preprint arXiv:1512.07199*, 2015.

[CSB16] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Mobile cloud business process management system for the internet of things: a survey. *ACM Computing Surveys*, 49(4):70, 2016.

[CSOL15] Sejin Chun, Seungmin Seo, Byungkook Oh, and Kyong-Ho Lee. Semantic description, discovery and integration for the internet of things. In *IEEE Intern. Conference on Semantic Computing (ICSC)*, pages 272–275, Feb 2015.

[DBBM11] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 949–955. IEEE, 2011.

[DCMR12] Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. Knowledge-intensive processes: An overview of contemporary approaches. In *KiBP@ KR*, pages 33–47, 2012.

[Dey01] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[DFZ+15] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. Everything as a service (xaas) on the cloud: origins, current and future trends. In *IEEE 8th Intern. Conf. on Cloud Computing (CLOUD)*, pages 621–628. IEEE, 2015.

[DG09] Akshay Dabholkar and Aniruddha Gokhale. An approach to middleware specialization for cyber physical systems. In *29th IEEE Intern. Conf. on Distributed Computing Systems Workshops*, pages 73–79. IEEE, 2009.

[DGLLS09] Giuseppe De Giacomo, Yves Lespérance, Hector J Levesque, and Sebastian Sardina. Indigolog: A high-level programming language for embedded reasoning agents. In *Multi-Agent Programming:*, pages 31–72. Springer, 2009.

[DLGM+13] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[DLRM+13] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013.

[DLV12] Patricia Derler, Edward A Lee, and Alberto Sangiovanni Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[DMC14] Dulce Domingos, Francisco Martins, and Carlos C. Internet of Things Aware WS-BPEL Business Processes - Context Variables and Expected Exceptions. *J. UCS 20.8*, 20(8):1109–1129, 2014.

[DMLYE18] Beniamino Di Martino, Kuan-Ching Li, Laurence Tianruo Yang, and Antonio Esposito. *Trends and Strategic Researches in Internet of Everything*, pages 1–12. Springer Singapore, Singapore, 2018.

[DR09] Peter Dadam and Manfred Reichert. The adept project: a decade of research and development for robust and flexible process support. *Computer Science-Research and Development*, 23(2):81–97, 2009.

[dR12] Auke Jan de Roo. *Managing software complexity of adaptive systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2012.

[DRRM+09] Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Andreas Lanz, Rüdiger Pryss, Michael Predeschly, Jens Kolb, Linh Thao Ly, Martin Jurisch, Ulrich Kreher, et al. From adept to aristaflow bpm suite: a research vision has become reality. In *International Conference on Business Process Management*, pages 529–531. Springer, 2009.

[dRSA11] Arjan de Roo, Hasan Sozer, and Mehmet Aksit. Runtime verification of domain-specific models of physical characteristics in control software. In *Fifth Intern. Conf. on Secure Software Integration and Reliability Improvement (SSIRI)*, pages 41–50. IEEE, 2011.

[DRSA12] Arjan De Roo, Hasan Sözer, and Mehmet Akşit. Verification and analysis of domain-specific models of physical characteristics in embedded control software. *Information and software technology*, 54(12):1432–1453, 2012.

[dRSA14] Arjan de Roo, Hasan Sözer, and Mehmet Akşit. Composing domain-specific physical models with general-purpose software modules in embedded control software. *Software & Systems Modeling*, 13(1):55–81, 2014.

[DS99] Axel Daneels and Wayne Salter. What is scada? In *International Conference on Accelerator and Large Experimental Physics Control Systems*, 1999.

[DTB+15] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, and Kurt Geihs. A resource oriented integration architecture for the Internet of Things: A business process perspective. *Pervasive and Mobile Computing*, 20:145–159, 2015.

[DTRE11] Kashif Dar, Amirhosein Taherkordi, Romain Rouvoy, and Frank Eliassen. Adaptable service composition for very-large-scale internet of things systems. In *Proceedings of the 8th Middleware Doctoral Symposium*, page 2. ACM, 2011.

[DVdATH05] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005.

[Erl05] Thomas Erl. Service-oriented architecture (soa): concepts, technology, and design, 2005.

[Fer99] Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[FLSK13] Karol Furdik, Gabriel Lukac, Tomas Sabol, and Peter Kostelnik. The network architecture designed for an adaptable iot-based smart office solution. *International Journal of Computer Networks and Communications Security*, 1(6):216–224, 2013.

[FP10] Marc Eduard Frîncu and Dana Petcu. Osyris: a nature inspired workflow engine for service oriented environments. *Scalable Computing: Practice and Experience*, 11(1):81–97, 2010.

[Fri11] Marc Eduard Frincu. D-osyris: A self-healing distributed workflow engine. In *Int. Symposium Parallel and Distributed Computing*, pages 215–222, 2011.

[FVH18] Christian Friedow, Maximilian Völker, and Marcin Hewelt. Integrating iot devices into business processes. In *International Conference on Advanced Information Systems Engineering*, pages 265–277. Springer, 2018.

[Gam95] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[GBF85] Andrew Goldenberg, Beno Benhabib, and Robert Fenton. A complete generalized solution to the inverse kinematics of robots. *IEEE Journal on Robotics and Automation*, 1(1):14–20, 1985.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GCB07] Mike Graves, Adam Constabaris, and Dan Brickley. Foaf: Connecting people on the semantic web. *Cataloging & classification quarterly*, 43(3-4):191–202, 2007.

[GCFP10] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, 9(2):18–26, 2010.

[GCFP11] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Implicit interaction design for pervasive workflows. *Personal Ubiquitous Comput.*, 15(4):399–408, April 2011.

[GdV98] Paul Grefen and Remmert Remmerts de Vries. A reference architecture for workflow management systems. *Data & Knowledge Engineering*, 27(1):31 – 57, 1998.

[GEPF11] Nils Glombitza, Sebastian Ebers, Dennis Pfisterer, and Stefan Fischer. Using bpel to realize business processes for an internet of things. In *International Conference on Ad-Hoc Networks and Wireless*, pages 294–307. Springer, 2011.

[GGAAPE+11] A Gonzalez-Garcia, A Alvarez-Alvarez, Jordán Pascual-Espada, Oscar Sanjuan-Martinez, Juan Manuel Cueva Lovelle, and B Cristina Pelayo G-Bustelo. Introduction to devices orchestration in internet of things using sbpmn. *International Journal of Interactive Multimedia and Artificial Intelligence*, 1(4), 2011.

[GGBG13] Levent Gurgen, Ozan Gunalp, Yazid Benazzouz, and Mathieu Gallissot. Self-aware cyber-physical systems and applications in smart buildings and cities. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1149–1154, San Jose, CA, USA, 2013. EDA Consortium.

[Gie16] Holger Giese. Formal models and analysis for self-adaptive cyber-physical systems. In *International Workshop on Formal Aspects of Component Software*, pages 3–9. Springer, 2016.

[GIM11] Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: Rest or ws-*? a developers perspective. In *Intern. Conf. on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337. Springer, 2011.

[GKGK16] Imen Graja, Slim Kallel, Nawal Guermouche, and Ahmed Hadj Kacem. Bpmn4cps: A bpmn extension for modeling cyber-physical systems. In *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 152–157, June 2016.

[GLR+15] Philipp Gerbert, Markus Lorenz, Michael Rüßmann, Manuela Waldner, Jam Justus, Pascal Engel, and Michael Harnisch. Industry 4.0-the future of productivity and growth in manufacturing industries. *The Boston Consulting Group*, 2015.

[GPGV14] Volkan Gunes, Steffen Peter, Tony Givargis, and Frank Vahid. A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Transactions on Internet & Information Systems*, 8(12), 2014.

[GR92] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[GSRU07] Debanjan Ghosh, Raj Sharman, H Raghav Rao, and Shambhu Upadhyaya. Self-healing systems–survey and synthesis. *Decision support systems*, 42(4):2164–2185, 2007.

[HCLB15] Robert R Harmon, Enrique G Castro-Leon, and Sandhiprakash Bhide. Smart cities and the internet of things. In *Management of Engineering and Technology (PICMET), 2015 Portland International Conference on*, pages 485–494. IEEE, 2015.

[HDG+06] Matthew Horridge, Nick Drummond, John Goodwin, Alan L Rector, Robert Stevens, and Hai Wang. The manchester owl syntax. In *OWLed*, volume 216, 2006.

[HHGR06] Gregory Hackmann, Mart Haitjema, Christopher Gill, and Gruia-Catalin Roman. Sliver: A bpel workflow process execution engine for mobile devices. In *ICSOC*, volume 4294, pages 503–508. Springer, 2006.

[HJS89] Peter Huber, Kurt Jensen, and Robert M Shapiro. Hierarchies in coloured petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 313–341. Springer, 1989.

[HMW13] Nico Herzberg, Andreas Meyer, and Mathias Weske. An Event Processing Platform for Business Process Management. *17th IEEE International Enterprise Distributed Object Computing Conference*, pages 107–116, 2013.

[HO13] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6, May 2013.

[HRR+08] Dirk Habich, Sebastian Richly, Andreas Ruempel, Wolfgang Buecke, and Steffen Preissler. Open service process platform 2.0. In *IEEE Congress on Services-Part I*, pages 152–159. IEEE, 2008.

[HRZ15] Andreas Holzinger, Carsten Röcker, and Martina Ziefle. From smart health to smart hospitals. In *Smart health*, pages 1–20. Springer, 2015.

[HSDV13] Philipp Hoenisch, Stefan Schulte, Schahram Dustdar, and Srikumar Venugopal. Self-adaptive resource allocation for elastic process execution. In *IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 220–227. IEEE, 2013.

[HSK+16] Steffen Huber, Ronny Seiger, André Kühnert, Vasileios Theodorou, and Thomas Schlegel. Goal-based semantic queries for dynamic processes in the internet of things. *International Journal of Semantic Computing*, 10(02):269–293, 2016.

[HSKS16a] Steffen Huber, Ronny Seiger, André Kühnert, and Thomas Schlegel. A context-adaptive workflow engine for humans, things and services. In *Proc. of the ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, pages 285–288, New York, NY, USA, 2016. ACM.

[HSKS16b] Steffen Huber, Ronny Seiger, André Kühnert, and Thomas Schlegel. Using semantic queries to enable dynamic service invocation for processes in the internet of things. In *IEEE Intern. Conference on Semantic Computing (ICSC)*, pages 214–221, Feb 2016.

[Hub18] Steffen Huber. *Goal-based Workflow Adaptation for Role-based Resources in the Internet of Things*. PhD thesis, 2018.

[HWS+16] Pascal Hirmer, Matthias Wieland, Holger Schwarz, Bernhard Mitschang, Uwe Breitenbücher, Santiago Gómez Sáez, and Frank Leymann. Situation recognition and handling based on executing situation templates and situation-aware workflows. *Computing*, pages 1–19, 2016.

[IBM05] IBM. An architectural blueprint for autonomic computing. Technical report, IBM, June 2005.

[Jaz14] Nasser Jazdi. Cyber physical systems in the context of industry 4.0. In *IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4. IEEE, 2014.

[JDK15] Dávid Juhász, László Domoszlai, and Barnabás Králik. Rea: Workflows for cyber-physical systems. In *Central European Functional Programming School*, pages 479–506. Springer, 2015.

[JGVDSJ14] Niels Joncheere, Sebastian Günther, Ragnhild Van Der Straeten, and Viviane Jonckers. Improving workflow modularity using a concern-specific layer on top of unify. *Science of Computer Programming*, 87:62–94, 2014.

[JHA+13] Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, and R Horrop. The spring framework-reference documentation, 2.0. 5, 2013.

[JKM+17] Christian Janiesch, Agnes Koschmider, Massimo Mecella, Barbara Weber, Andrea Burattin, Claudio Di Ciccio, Avigdor Gal, Udo Kannengiesser, Felix Mannhardt, Jan Mendling, et al. The internet-of-things meets business process management: Mutual benefits and challenges. *arXiv preprint arXiv:1709.03628*, 2017.

[Joh02] Corinne N Johnson. The benefits fo pdca. *Quality Progress*, 35(5):120, 2002.

[JROK11] Jae-Yoon Jung, Pablo Rosales, Kyuhyup Oh, and Kyuri Kim. eduflow: An event-driven ubiquitous flow management system. In *International Conference on Business Process Management*, pages 427–432. Springer, 2011.

[KA10] Jan Kleissl and Yuvraj Agarwal. Cyber-physical energy systems: Focus on smart buildings. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 749–754. IEEE, 2010.

[KAK16] Meesun Kim, Hyun Ahn, and Kwanghoon Pio Kim. Process-aware internet of things: A conceptual extension of the internet of things framework and architecture. *KSII Transactions on Internet & Information Systems*, 10(8), 2016.

[KBG13] Dmitry G. Korzun, Sergey I. Balandin, and Andrei V. Gurtov. Deployment of smart spaces in internet of things: Overview of the design challenges. In Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networking*, pages 48–59, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KGC+12] Pratyush Kumar, Dip Goswami, Samarjit Chakraborty, Anuradha Annaswamy, Kai Lampka, and Lothar Thiele. A hybrid approach to cyber-physical systems verification. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 688–696, New York, NY, USA, 2012. ACM.

[KK12] Falko Koetter and Monika Kochanowski. *Goal-Oriented Model-Driven Business Process Monitoring Using ProGoalML*, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[KKL+05] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. Ws-bpel extension for people–bpel4people. *Joint white paper, IBM and SAP*, 183:184, 2005.

[KKMZ17] Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun (eds.) Zhu. *Self-Aware Computing Systems*. Springer, 2017.

[KKS11] Romina Kühn, Christine Keller, and Thomas Schlegel. A context taxonomy supporting public system design. In *Proceedings of the 1st International Workshop on Model-based Interactive Ubiquitous Systems, EICS, to appear*, 2011.

[KKSF10] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[KLB+17] Samuel Kounev, Peter Lewis, Kirstie L Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger

Giese, Sebastian Götz, et al. The notion of self-aware computing. In *Self-Aware Computing Systems*, pages 3–16. Springer, 2017.

[KLG+14] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. A metamodel family for role-based modeling and programming languages. In *International Conference on Software Language Engineering*, pages 141–160. Springer, 2014.

[KLW11] Henning Kagermann, Wolf-Dieter Lukas, and Wolfgang Wahlster. Industrie 4.0: Mit dem internet der dinge auf dem weg zur 4. industriellen revolution. *VDI nachrichten*, 13:11, 2011.

[KM07] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering, 2007. FOSE'07*, pages 259–268. IEEE, 2007.

[Kop13] Hermann Kopetz. System-of-systems complexity. *arXiv preprint arXiv:1311.3629*, 2013.

[Kou11] Samuel Kounev. Engineering of self-aware it systems and services: State-of-the-art and research challenges. *Computer Performance Engineering*, pages 10–13, 2011.

[KP15] Attila Kertesz and Tamas Pflanzner. Towards enabling scientific workflows for the future internet of things. In *International Internet of Things Summit*, pages 399–408. Springer, 2015.

[KPGV03] Gail Kaiser, Janak Parekh, Philip Gross, and Giuseppe Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *Proceedings of the Autonomic Computing Workshop*, pages 22–30. IEEE, 2003.

[KSKP11] Nikos Kefalakis, John Soldatos, Nikolaos Konstantinou, and Neeli R Prasad. Apdl: A reference xml schema for process-centered definition of rfid solutions. *Journal of Systems and Software*, 84(7):1244–1259, 2011.

[LBK15] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[Lec09] Jens Lechtenbörger. Two-phase commit protocol. In *Encyclopedia of Database Systems*, pages 3209–3213. Springer, 2009.

[Lee08] Edward A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369, 2008.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[LM06] Stefan-Helmut Leitner and Wolfgang Mahnke. Opc ua–service-oriented architecture for industrial applications. *ABB Corporate Research Center*, 2006.

[LM16] Luís Lopes and Francisco Martins. A safe-by-design programming language for wireless sensor networks. *Journal of Systems Architecture*, 63:16–32, 2016.

[LMM15] Francesco Leotta, Massimo Mecella, and Jan Mendling. Applying process mining to smart spaces: Perspectives and research challenges. In *Advanced Information Systems Engineering Workshops*, pages 298–304. Springer, 2015.

[Lok03] Seng Wai Loke. Service-oriented device ecology workflows. In *International Conference on Service-Oriented Computing*, pages 559–574. Springer, 2003.

[LPR98] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus, 1998.

[LPS+09] Kevin Lee, Norman W Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. Adaptive workflow processing and execution in pegasus. *Concurrency and Computation: Practice and Experience*, 21(16):1965–1981, 2009.

[LRD10] Andreas Lanz, Manfred Reichert, and Peter Dadam. Robust and flexible error handling in the aristaflow bpm suite. In *Forum at the Conference on Advanced Information Systems Engineering (CAiSE)*, pages 174–189. Springer, 2010.

[LS16] Edward Ashford Lee and Sanjit A Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[Luc02] David Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.

[Mar17] Andrea Marrella. What automated planning can do for business process management. *arXiv preprint arXiv:1709.10482*, 2017.

[MBBF17] Jan Mendling, Bart Baesens, Abraham Bernstein, and Michael Fellmann. Challenges of smart business process management: An introduction to the special issue. *Decision Support Systems*, 2017.

[MCS16] Jakob Mass, Chii Chang, and Satish Narayana Srirama. Wiseware: A device-to-device-based business process management system for industrial internet of things. In *IEEE Intern. Conf. on Internet of Things (iThings), Green Computing and Communications (GreenCom), Cyber, Physical and Social Computing (CPSCom) and Smart Data (SmartData)*, pages 269–275. IEEE, 2016.

[MD17] Francisco Martins and Dulce Domingos. Modelling iot behaviour within bpmn business processes. *Procedia Computer Science*, 121:1014–1022, 2017.

[MDCM17] Giovanni Meroni, Claudio Di Ciccio, and Jan Mendling. An artifact-driven approach to monitor business processes through real-world objects. In *International Conference on Service-Oriented Computing*, pages 297–313. Springer, 2017.

[Mez16] Jan Meznaric. *Extending BPMN for integration of internet of things devices with process-driven applications*. PhD thesis, 2016.

[MGR04] Robert Müller, Ulrike Greiner, and Erhard Rahm. Agentwork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51(2):223–256, 2004.

[MHW17] Sankalita Mandal, Marcin Hewelt, and Mathias Weske. A framework for integrating real-world events and business processes in an iot environment. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 194–212. Springer, 2017.

[MJPL12] Luis E Gonzalez Moctezuma, Jani Jokinen, Corina Postelnicu, and Jose L Martinez Lastra. Retrofitting a factory automation system to address market needs and societal changes. In *10th IEEE Int. Conf. on Industrial Informatics (INDIN)*, pages 413–418. IEEE, 2012.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[MM05] Frederic Montagut and Refik Molva. Enabling pervasive execution of workflows. In *Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing*, pages 10 pp.–, 2005.

[MM17] Andrea Marrella and Massimo Mecella. Cognitive business process management for adaptive cyber-physical processes. In *Int. Conf. on Business Process Management*, pages 429–439. Springer, 2017.

[MMG08] Frederic Montagut, Refik Molva, and Silvan Tecumseh Golega. The pervasive workflow: A decentralized workflow system supporting long-running transactions. *IEEE Transactions on Systems, Man, and Cybernetics*, 38(3):319–333, May 2008.

[MMHS15] Andrea Marrella, Massimo Mecella, Patris Halapuu, and Sebastian Sardina. Automated process adaptation in cyber-physical domains with the smartpm system (short paper). In *IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 59–64. IEEE, 2015.

[MMP06] Stefano Modafferi, Enrico Mussi, and Barbara Pernici. Sh-bpel: a self-healing plug-in for ws-bpel engines. In *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 48–53. ACM, 2006.

[MMS14] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. Smartpm: an adaptive process management system through situation calculus, indigolog, and classical planning. In *Principles of Knowledge Representation and Reasoning*, pages 1–10. AAAI Press, 2014.

[MMS16] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. Intelligent process adaptation in the smartpm system. *ACM Trans. Intell. Syst. Technol.*, 8(2):25:1–25:43, November 2016.

[MMS17] Andrea Marrella, Massimo Mecella, and Sebastian Sardiña. Supporting adaptiveness of cyber-physical processes through action-based formalisms. *AI Communications*, (Preprint):1–28, 2017.

[Mod11] Business Process Model. Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*, pages 22–31, 2011.

[Mon13] Olivier Monnier. A smarter grid with the internet of things. *Texas Instruments*, 2013.

[Mon14] László Monostori. Cyber-physical production systems: roots, expectations and r&d challenges. *Procedia Cirp*, 17:9–13, 2014.

[MPMR16] Simon Mayer, Dominic Plangger, Florian Michahelles, and Simon Rothfuss. Ubermanufacturing: A goal-driven collaborative industrial manufacturing marketplace. In *Proc. of 6th Int. Conf. on the Internet of Things*, pages 111–119, New York, NY, USA, 2016. ACM.

[MPO+17] Luca Mottola, Gian Pietro Picco, Felix Jonathan Opperman, Joakim Eriksson, Niclas Finne, Harald Fuchs, Andrea Gaglione, Stamatis Karnouskos, Patricio Montero, Nina Oertel, et al. makesense: Simplifying the integration of wireless sensor networks into business processes. *IEEE Trans. on Software Engineering*, 2017.

[MRH15] Sonja Meyer, Andreas Ruppen, and Lorenz Hilty. The things of the internet of things in bpmn. In *Advanced Information Systems Engineering Workshops*, pages 285–297, 2015.

[MRM13] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: Integrating IoT devices as business process resources. *Lecture Notes in Computer Science*, 7908 LNCS:84–98, 2013.

[MRM14] Juergen Mangler and Stefanie Rinderle-Ma. Cpee - cloud process execution engine. In *BPM (Demos)'14*, pages 51–51, 2014.

[MS17] Henry Muccini and Mohammad Sharaf. Caps: a tool for architecting situational-aware cyber-physical systems. In *IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 286–289. IEEE, 2017.

[MSDPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[MSW16] Henry Muccini, Mohammad Sharaf, and Danny Weyns. Self-adaptation for cyber-physical systems: A systematic literature review. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '16, pages 75–81, New York, NY, USA, 2016. ACM.

[MVKM16] Simon Mayer, Ruben Verborgh, Matthias Kovatsch, and Friedemann Mattern. Smart configuration of smart environments. *IEEE Transactions on Automation Science and Engineering*, 13(3):1247–1255, 2016.

[MVSA16] Mohammad Masdari, Sima ValiKardan, Zahra Shahi, and Sonay Imani Azar. Towards workflow scheduling in cloud computing. *J. Netw. Comput. Appl.*, 66(C):64–82, May 2016.

[Nac12] Nachwuchsforschergruppe VICCI. Vicci: Visual and interactive cyber-physical systems control and integration, 2012.

[NND+17] Matteo Nardelli, Stefan Nastic, Schahram Dustdar, Massimo Villari, and Rajiv Ranjan. Osmotic flow: Osmotic computing+ iot workflow. *IEEE Cloud Computing*, 4(2):68–75, 2017.

[NSKS14] Florian Niebling, Daniel Schropp, Romina Kühn, and Thomas Schlegel. Model-based multi-touch gesture interaction for diagram editors. In *International Conference on Human-Computer Interaction*, pages 121–130. Springer, 2014.

[NSS14] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[NSW13] Fernando Niroshinie, W Loke Seng, and Rahayu Wenny. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[OAS14] OASIS. Web Services Business Process Execution Language (WS-BPEL), 2014.

[OCEP13] Karolyne Oliveira, Jaelson Castro, Sergio España, and Oscar Pastor. Multi-level autonomic business process management. *Enterprise, Business-Process and Information Systems Modeling*, pages 184–198, 2013.

[OG17] Roy Oberhauser and Gregor Grambow. Towards autonomically-capable processes: A vision and potentially supportive methods. In *Advances in Intelligent Process-Aware Information Systems*, pages 79–125. Springer, 2017.

[OHG17] Julius Ollesch, Marc Hesenius, and Volker Gruhn. Engineering events in cps: experiences and lessons learned. In *Proceedings of the 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 3–9. IEEE Press, 2017.

[Omg08] QVT Omg. Meta object facility (mof) 2.0 query/view/transformation specification. *Final Adopted Specification (November 2005)*, 2008.

[Pap03] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proc. of the Fourth Intern. Conf. on Web Information Systems Engineering*, pages 3–12. IEEE, 2003.

[PD11] Harald Psaier and Schahram Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, 2011.

[PDB+08] Carlos Pedrinaci, John Domingue, Christian Brelage, Tammo Van Lessen, Dimka Karastoyanova, and Frank Leymann. Semantic business process management: Scaling up the management of business processes. In *IEEE International Conference on Semantic Computing*, pages 546–553. IEEE, 2008.

[Pet81] James L Peterson. Petri net theory and the modeling of systems. 1981.

[PKGV06] Janak Parekh, Gail Kaiser, Philip Gross, and Giuseppe Valetto. Retrofitting autonomic capabilities onto legacy systems. *Cluster Computing*, 9(2):141–159, 2006.

[PLM16] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. Cyber-physical objects as key elements for a smart cyber-city. In *Management of Cyber Physical Objects in the Future Internet of Things*, pages 31–49. Springer, 2016.

[PLM17] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies*, 28(1), 2017.

[PNC+14] Milan Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *Mobile-edge Computing (MEC) industry initiative*, 2014.

[PRBA15] Rüdiger Pryss, Manfred Reichert, Alexander Bachmeier, and Johann Albach. Bpm to go: Supporting business processes in a mobile and sensing world. 2015.

[PRK+14] Christian Piechnick, Sebastian Richly, Thomas Kühn, Sebastian Götz, Georg Püschel, and Uwe Aßmann. Contextpoint: An architecture for extrinsic meta-adaptation in smart environments, 2014.

[PRS+13] Tao Peng, Marco Ronchetti, Jovan Stevovic, Annamaria Chiasera, and Giampaolo Armellin. Business process assignment and execution from cloud to mobile. In *International Conference on Business Process Management*, pages 264–276. Springer, 2013.

[PSS12] Georg Püschel, Ronny Seiger, and Thomas Schlegel. Test modeling for context-aware ubiquitous applications with feature petri nets. In *Proc. Workshop Model-based Interactive Ubiquitous Systems (MODIQUITOUS)*, 2012.

[PTDL07] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, (11):38–45, 2007.

[PTKR10] Rüdiger Pryss, Julian Tiedeken, Ulrich Kreher, and Manfred Reichert. Towards flexible process support on mobile devices. In *Forum at the Conference on Advanced Information Systems Engineering (CAiSE)*, pages 150–165. Springer, 2010.

[PWLK03] Sang Hyun Park, So Hee Won, Jong Bong Lee, and Sung Woo Kim. Smart home–digitally engineered domestic life. *Personal and Ubiquitous Computing*, 7(3-4):189–196, 2003.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[RBA08] Sebastian Richly, Wolfgang Buecke, and Uwe Aßmann. A bdi-based reflective infrastructure for dynamic workflows. In *2008 12th Enterprise Distributed Object Computing Conference Workshops*, pages 112–119, Sept 2008.

[RBD+09] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems*, pages 164–182. Springer, 2009.

[RBK+12] Heorhi Raik, Antonio Bucchiarone, Nawaz Khurshid, Annapaola Marconi, and Marco Pistore. Astro-captevo: Dynamic context-aware adaptation for service-based systems. In *IEEE Eighth World Congress on Services (SERVICES)*, pages 385–392. IEEE, 2012.

[RG+95] Anand S Rao, Michael P Georgeff, et al. Bdi agents: from theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[RPTC15] Alessandro Ricci, Michele Piunti, Luca Tummolini, and Cristiano Castelfranchi. The mirror world: Preparing for mixed-reality living. *IEEE Pervasive Computing*, 14(2):60–63, 2015.

[RRD03] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. Adept workflow management system. In Wil M. P. van der Aalst and Mathias Weske, editors, *Business Process Management*, pages 370–379, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[RRKD05] Manfred Reichert, Stefanie Rinderle, Ulrich Kreher, and Peter Dadam. Adaptive process management with adept2. In *Proceedings. 21st International Conference on Data Engineering,. ICDE*, pages 1113–1114. IEEE, 2005.

[RS16] Daniel Ritter and Jan Sosulski. Exception handling in message-based integration systems and modeling using bpmn. *International Journal of Cooperative Information Systems*, 25(02):1650004, 2016.

[RSA10] Sebastian Richly, Sandro Schmidt, and Uwe Aßmann. A Semantic-BDI-based approach to realize cooperative, reflexive workflows. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pages 1680–1685, 2010.

[RSI+17] Michele Ruta, Floriano Scioscia, Saverio Ieva, Giuseppe Loseto, Filippo Gramegna, and Agnese Pinto. Knowledge discovery and sharing in the iot: the physical semantic web vision. In *Proc. of the Symposium on Applied Computing*, pages 492–498. ACM, 2017.

[RSS13] Hajo A Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling–an academic dream or the future for bpm? In *Business Process Management*, pages 307–322. Springer, 2013.

[RvWLB15] Roland Rosen, Georg von Wichert, George Lo, and Kurt D Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48(3):567–572, 2015.

[RW15] F. J. Riggins and S. F. Wamba. Research directions on the adoption, usage, and impact of the internet of things through the use of big data analytics. In *2015 48th Hawaii International Conference on System Sciences*, pages 1531–1540, Jan 2015.

[RWE15] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data.* " O'Reilly Media, Inc.", 2015.

[RWRW05] Stefanie Rinderle, Barbara Weber, Manfred Reichert, and Werner Wild. Integrating process learning and process evolution-a semantics based approach. *Business Process Management*, 3649:252–267, 2005.

[SAEJ18] Stefan Schönig, Ana Paula Aires, Andreas Ermer, and Stefan Jablonski. Workflow support in wearable production information systems. In *International Conference on Advanced Information Systems Engineering*, pages 235–243. Springer, 2018.

[SASS15] Ronny Seiger, Bashar Altakrouri, Andreas Schrader, and Thomas Schlegel, editors. *Proceedings of the 1st Workshop on Large-scale and Model-based Interactive Systems: Approaches and Challenges (LMIS 2015)*, number 1380 in CEUR Workshop Proceedings, Aachen, 2015.

[SBW99] Clemens Szyperski, Jan Bosch, and Wolfgang Weck. Component-oriented programming. In *Object-oriented technology ecoop99 workshop reader*, pages 184–192. Springer, 1999.

[SCA+17] Kunal Suri, Juan Cadavid, Mauricio Alferez, Saadia Dhouib, and Sara Tucci-Piergiovanni. Modeling business motivation and underlying processes for rami 4.0-aligned cyber-physical production systems. In *22nd IEEE International Conference on Emerging Technologies And Factory Automation, At Limassol, Cyprus*, 2017.

[Sch08] Thomas Schlegel. *Laufzeit-Modellierung objektorientierter interaktiver Prozesse in der Produktion.* PhD thesis, 2008.

[Sch09] Thomas Schlegel. Object-oriented interactive processes in decentralized production systems. In *Human Interface and the Management of Information. Designing Information Environments*, volume 5617 of *Lecture Notes in Computer Science*, pages 296–305. Springer Berlin Heidelberg, 2009.

[Sch13] August-Wilhelm Scheer. *ARISvom Geschäftsprozess zum Anwendungssystem.* Springer-Verlag, 2013.

[SDA+15] Karolj Skala, Davor Davidovic, Enis Afgan, Ivan Sovic, and Zorislav Sojat. Scalable distributed computing hierarchy: Cloud, fog and dew computing. *Open Journal of Cloud Computing (OJCC)*, 2(1):16–24, 2015.

[SDFGB09] Hong Sun, Vincenzo De Florio, Ning Gui, and Chris Blondia. Promises and challenges of ambient assisted living systems. In *Sixth International Conference on Information Technology: New Generations, ITNG'09*, pages 1201–1207. Ieee, 2009.

[SECP13] Gunar Schirner, Deniz Erdogmus, Kaushik Chowdhury, and Taskin Padir. The future of human-in-the-loop cyber-physical systems. *Computer*, (1):36–45, 2013.

[Sei15] Ronny Seiger. Modelling and execution of consistent and distributed workflows for cyber-physical systems. In *Business Process Management (Doctoral Consortium)*, 2015.

[SG07] H. Sahin and L. Guvenc. Household robotics: autonomous devices for vacuuming and lawn mowing. *Control Systems, IEEE*, 27(2):20–96, April 2007.

[SGCG17] Kunal Suri, Walid Gaaloul, Arnaud Cuccuru, and Sebastien Gerard. Semantic framework for internet of things-aware business process development. In *IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 214–219. IEEE, 2017.

[SGCG18] Kunal Suri, Walid Gaaloul, Arnaud Cuccuru, and Sebastien Gerard. Semantic framework for energy-aware resource management of iot in business processes. *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*, 8(1):21–43, 2018.

[SGLW08] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing, SUTC'08.*, pages 1–9. IEEE, 2008.

[SGS11] Ronny Seiger, Stephan Groß, and Alexander Schill. Seccsie: A secure cloud storage integrator for enterprises. In *2011 IEEE 13th Conference on Commerce and Enterprise Computing*, pages 252–255, Sept 2011.

[SH05] Roy Sterritt and Mike Hinchey. Apoptosis and self-destruct: A contribution to autonomic agents? In Michael G. Hinchey, James L. Rash, Walter F. Truszkowski, and Christopher A. Rouff, editors, *Formal Approaches to Agent-Based Systems*, pages 262–270, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[SHA17] Ronny Seiger, Stefan Herrmann, and Uwe Aßmann. Self-healing for distributed workflows in the internet of things. In *IEEE Int. Conference on Software Architecture (ICSA) Workshops*, 2017.

[SHA18a] Ronny Seiger, Peter Heisig, and Uwe Aßmann. Retrofitting of workflow management systems with self-x capabilities for internet of things. In *BP-Meet-IoT Workshop, Int. Conference on Business Process Management (BPM) Workshops*, 2018.

[SHA18b] Ronny Seiger, Stefen Huber, and Uwe Aßmann. A case study for workflow-based automation in the internet of things. In *IEEE Int. Conference on Software Architecture (ICSA) Companion*, 2018.

[SHH+14] Stefan Schulte, Philipp Hoenisch, Christoph Hochreiner, Schahram Dustdar, Matthias Klusch, and Dieter Schuller. Towards process support for cloud manufacturing. In *IEEE 18th Int. Enterprise Distributed Object Computing Conf. (EDOC)*, pages 142–149, 2014.

[SHH17] Ronny Seiger, Steffen Huber, and Peter Heisig. Proteus++: A self-managed iot workflow engine with dynamic service discovery. In *9th Central European Workshop on Services and their Composition (ZEUS)*, 2017.

[SHHA16] Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. *Enabling Self-adaptive Workflows for Cyber-physical Systems*, pages 3–17. Springer International Publishing, 2016.

[SHHA17] Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software & Systems Modeling*, Nov 2017.

[SHS15] Ronny Seiger, Steffen Huber, and Thomas Schlegel. *PROtEUS: An Integrated System for Process Execution in Cyber-Physical Systems*, pages 265–280. Springer International Publishing, 2015.

[SHS16] Ronny Seiger, Steffen Huber, and Thomas Schlegel. Toward an execution system for self-healing workflows in cyber-physical systems. *Software & Systems Modeling*, pages 1–22, 2016.

[SHS17] Ronny Seiger, Steffen Huber, and Thomas Schlegel. An execution system for self-healing workflows in cyber-physical systems. In *Software Engineering 2017*, number Lecture Notes in Informatics (LNI), pages 75–76. Gesellschaft für Informatik, 2017.

[SHVD12] Stefan Schulte, Philipp Hoenisch, Srikumar Venugopal, and Schahram Dustdar. Introducing the vienna platform for elastic processes. In *International Conference on Service-Oriented Computing*, pages 179–190. Springer, 2012.

[SJV+15] Stefan Schulte, Christian Janiesch, Srikumar Venugopal, Ingo Weber, and Philipp Hoenisch. Elastic business process management: State of the art and open challenges for bpm in the cloud. *Future Generation Computer Systems*, 46:36–50, 2015.

[SKGA17] Ronny Seiger, Mandy Korzetz, Maria Gohlke, and Uwe Aßmann. Mixed reality cyber-physical systems control and workflow composition. In *Proc. of the 16th Int. Conf. on Mobile and Ubiquitous Multimedia*, MUM '17. ACM, 2017.

[SKNS13] Ronny Seiger, Christine Keller, Florian Niebling, and Thomas Schlegel. Modelling complex and flexible processes for smart cyber-physical environments. In *Proceedings of 25th European Modeling and Simulation Symposium (EMSS)*, pages 73–82, September 2013.

[SKNS15] Ronny Seiger, Christine Keller, Florian Niebling, and Thomas Schlegel. Modelling complex and flexible processes for smart cyber-physical environments. *Journal of Computational Science*, 10:137 – 148, 2015.

[SLI08] Sucha Smanchat, Sea Ling, and Maria Indrawan. A survey on context-aware workflow adaptations. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 414–417. ACM, 2008.

[SLSS16] Ronny Seiger, Diana Lemme, Susann Struwe, and Thomas Schlegel. An interactive mobile control center for cyber-physical systems. In *Proc. of the Int. Joint Conf. on Pervasive and Ubiquitous Computing: Adjunct*, pages 193–196, New York, NY, USA, 2016. ACM.

[SNK+15] Ronny Seiger, Florian Niebling, Mandy Korzetz, Tobias Nicolai, and Thomas Schlegel. A framework for rapid prototyping of multimodal interaction concepts. *Large-scale and Model-based Interactive Systems*, pages 21–28, 2015.

[SNS14a] Ronny Seiger, Tobias Nicolai, and Thomas Schlegel. A framework for controlling robots via brain-computer interfaces. In Andreas Butz, Michael Koch, and Johann Schlichter, editors, *Mensch & Computer 2014 - Workshopband*, pages 003–006, Berlin, 2014. De Gruyter Oldenbourg.

[SNS14b] Ronny Seiger, Florian Niebling, and Thomas Schlegel. A distributed execution environment enabling resilient processes for ubiquitous systems. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 220–223, March 2014.

[SS12] Erwin Schoitsch and Amund Skavhaug. Introduction to the ercim/ewics cyberphysical systems workshop 2012. In *International Conference on Computer Safety, Reliability, and Security*, pages 343–346. Springer, 2012.

[SSAS15] Ronny Seiger, Christoph Seidl, Uwe Aßmann, and Thomas Schlegel. A capability-based framework for programming small domestic service robots. In *Proc. of the 2015 Joint MORSE/VAO Workshop.*, pages 49–54, New York, NY, USA, 2015. ACM.

[SSKK15] Thomas Schlegel, Ronny Seiger, Christine Keller, and Romina Kühn. Model-based interactive ubiquitous systems (modiquitous). In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 296–297, New York, NY, USA, 2015. ACM.

[SSMS14] Ronny Seiger, Susann Struwe, Sandra Matthes, and Thomas Schlegel. A resilient interaction concept for process management on tabletops for cyber-physical systems. In *Human Interface and the Management of Information. Information and Knowledge in Applications and Services*, pages 347–358. Springer, 2014.

[SSOK13] C Timurhan Sungur, Patrik Spiess, Nina Oertel, and Oliver Kopp. Extending BPMN for Wireless Sensor Networks. *2013 IEEE 15th Conference on Business Informatics*, pages 109–116, 2013.

[STA05] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. *Process-Aware Information Systems*, pages 119–146, 2005.

[Sto15] André Stork. Visual computing challenges of advanced manufacturing and industrie 4.0 [guest editors' introduction]. *IEEE Computer Graphics and Applications*, 35(2):21–25, Mar 2015.

[SVDS12] Thomas Schlegel, Krešimir Vidačković, Sebastian Dusch, and Ronny Seiger. Management of interactive business processes in decentralized service infrastructures through event processing. *Journal of King Saud University - Computer and Information Sciences*, 24(2):137 – 144, 2012.

[SW14] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *Federated Conf. on Computer Science and Information Systems (FedCSIS)*, pages 1–8. IEEE, 2014.

[SWC+18] Rongjia Song, Ying Wang, Weiping Cui, Jan Vanthienen, and Lei Huang. Towards improving context interpretation in the iot paradigm: a solution to integrate context information in process models. In *2nd Int. Conf. on Management Engineering, Software Engineering and Service Sciences*, pages 223–228. ACM, 2018.

[SWYS11] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber-physical systems. In *International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2011.

[SYY07] Jun Shen, Yun Yang, and Jun Yan. A p2p based service flow system with advanced ontology-based service profiles. *Advanced Engineering Informatics*, 21(2):221 – 229, 2007.

[SZY18] Yuan Sun, Xingshe Zhou, and Gang Yang. Location sensitive multitask oriented service composition for cyber physical systems. *International Journal of Innovative Computing, Information and Control*, 14(3), 2018.

[SZZ14] Lukas Smirek, Gottfried Zimmermann, and Daniel Ziegler. Towards universally usable smart homes-how can myui, urc and openhab contribute to an adaptive user interface platform. In *IARIA Conference, Nice, France*, pages 29–38, 2014.

[Tal08] Carolyn Talcott. Cyber-physical systems and events. In *Software-Intensive Systems and New Computing Paradigms*, pages 101–115. Springer, 2008.

[TFR17] P. Tsoutsa, Panos Fitsilis, and Omiros Ragos. Role modeling of iot services in industry domains. In *Proc. of the Intern. Conf. on Management Engineering, Software Engineering and Service Sciences*, pages 290–295, New York, NY, USA, 2017. ACM.

[TGD+08] Feilong Tang, Minyi Guo, Mianxiong Dong, Minglu Li, and Hu Guan. Towards context-aware workflow management for ubiquitous computing. In *Intern. Conference on Embedded Software and Systems, 2008. ICESS'08*, pages 221–228. IEEE, 2008.

[TMS+12] Matthias Thoma, Sonja Meyer, Klaus Sperner, Stefan Meissner, and Torsten Braun. On iot-services: Survey, classification and enterprise integration. In *IEEE Intern. Conf. on Green Computing and Communications (GreenCom)*, pages 257–260. IEEE, 2012.

[TSD+12] Stefano Tranquillini, Patrik Spieß, Florian Daniel, Stamatis Karnouskos, Fabio Casati, Nina Oertel, Luca Mottola, Felix Oppermann, Gian Picco, Kay Römer, et al. Process-based design and integration of wireless sensor network applications. *Business Process Management*, pages 134–149, 2012.

[VDA96] Wil MP Van Der Aalst. Three good reasons for using a petri-net-based workflow management system. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC96)*, pages 179–201. Citeseer, 1996.

[VdA98] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[vdA00] Wil van der Aalst. Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries. *Information & Management*, 37(2):67 – 75, 2000.

[VDA13] Wil MP Van Der Aalst. Business process management: a comprehensive survey. *ISRN Software Engineering*, 2013, 2013.

[VDAADM+11] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *Intern. Conference on Business Process Management*, pages 169–194. Springer, 2011.

[vDAPS09] Wil MP van Der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, 23(2):99–113, 2009.

[vdAtH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.

[vDATHKB03] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[VGC+15] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *Computing Colombian Conference (10CCC), 2015 10th*, pages 583–590. IEEE, 2015.

[vZvDvdA18] Sebastiaan J van Zelst, Boudewijn F van Dongen, and Wil MP van der Aalst. Event stream-based process discovery using abstract representations. *Knowledge and Information Systems*, 54(2):407–435, 2018.

[WBJ08] Daniel Work, Alexandre Bayen, and Quinn Jacobson. Automotive cyber physical systems in the context of human mobility. In *National Workshop on high-confidence automotive cyber-physical systems*, pages 3–4, 2008.

[WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005.

[Web12] Jim Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 217–218. ACM, 2012.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[Wes12] Mathias Weske. *Business Process Management Architectures*, pages 333–371. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Whi05] Stephen White. Using bpmn to model a bpel process. *BPTrends*, 3(3):1–18, 2005.

[WN14] Jonas Westman and Mattias Nyberg. Environment-centric contracts for design of cyber-physical systems. In *Int. Conf. on Model Driven Engineering Languages and Systems*, pages 218–234. Springer, 2014.

[Wom11a] Andreas Wombacher. A-posteriori detection of sensor infrastructure errors in correlated sensor data and business workflows. In *Proc. of the 9th Intern. Conference on Business Process Management*, pages 329–344, Berlin, Heidelberg, 2011. Springer-Verlag.

[Wom11b] Andreas Wombacher. How physical objects and business workflows can be correlated. *Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011*, pages 226–233, 2011.

[WRWR05] Barbara Weber, Stefanie Rinderle, Werner Wild, and Manfred Reichert. *CCBR–Driven Business Process Evolution*, pages 610–624. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[WSBL15] Matthias Wieland, Holger Schwarz, Uwe Breitenbucher, and Frank Leymann. Towards situation-aware adaptive workflows: Sitopt–a general purpose situation-aware workflow management system. In *PerCom Workshops*, pages 32–37. IEEE, 2015.

[WSJ15] Roy Want, Bill N Schilit, and Scott Jenson. Enabling the internet of things. *Computer*, 48(1):28–35, 2015.

[WvdAD+06] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, Arthur HM ter Hofstede, and Nick Russell. On the suitability of bpmn for business process modelling. In *International conference on business process management*, pages 161–176. Springer, 2006.

[WZG+14] Matthias Weidlich, Holger Ziekow, Asaf Gal, Jan Mendling, and Mathias Weske. Optimizing event pattern matching using business process models. *Knowledge and Data Engineering, IEEE Transactions on*, 26(11):2759–2773, 2014.

[WZZ+14] Jiafu Wan, Daqiang Zhang, Shengjie Zhao, Laurence Yang, and Jaime Lloret. Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. *IEEE Communications Magazine*, 52(8):106–113, 2014.

[XRK08] Kun Xiao, Shangping Ren, and Kevin Kwiat. Retrofitting cyber physical systems for survivability through external coordination. In *Proc. of the 41st Annual Hawaii Intern. Conf. on System Sciences*, pages 465–465. IEEE, 2008.

[YBSD16] Alaaeddine Yousfi, Christine Bauer, Rajaa Saidi, and Anind K. Dey. ubpmn: A bpmn extension for modeling ubiquitous business processes. *Information and Software Technology*, 74:55 – 68, 2016.

[YCZM07] Zhixian Yan, Emilia Cimpian, Michal Zaremba, and Manuel Mazzara. Bpmo: Semantic business process modeling and wsmo extension. In *IEEE Intern. Conf. on Web Services, ICWS*, pages 1185–1186. IEEE, 2007.

[YGM01] Beverly Yang and Hector Garcia-Molina. Comparing hybrid peer-to-peer systems. In *27th International Conference on Very Large Data Bases (VLDB 2001)*, September 2001.

[YHBW17] Alaaeddine Yousfi, Marcin Hewelt, Christine Bauer, and Mathias Weske. Towards ubpmn-based patterns for modeling ubiquitous business processes. *IEEE Trans. on Industrial Informatics*, 2017.

[ZHKL10] Sonja Zaplata, Kristof Hamann, Kristian Kottke, and Winfried Lamersdorf. Flexible execution of distributed business processes based on process instance migration. *Journal of Systems Integration*, 1(3):3, 2010.

Acronyms

AAL Ambient Assisted Living. 22, 23, 32, 56, 85, 127, 241

ACID Atomicity, Consistency, Isolation, Durability. 100, 104, 141, 161, 162, 230

API Application Programming Interface. 84, 90, 124, 170, 293, 294

BPM Business Process Management. 15, 16, 19–21, 23, 25, 26, 30, 38, 41, 42, 44, 45, 47–49, 52, 53, 58, 63, 71, 72, 98, 111, 112, 165, 233, 237–239, 241–244, 246–248, 281

BPMN Business Process Model and Notation. 21, 23, 26–28, 46–53, 56, 58, 62, 67, 75, 80, 81, 84, 85, 95, 111–113, 120, 168, 214, 246, 281

BPMS Business Process Management System. 24, 28–30, 48, 57, 63, 65, 69, 71, 244, 247, 281

CEP Complex Event Processing. 46, 48, 50, 53, 55, 57, 58, 72, 75, 84–86, 88, 121, 122, 145, 178, 182, 217–219, 229, 231, 233, 238, 239, 244, 282

CPPS Cyber-physical Production Systems. 17, 52

CPS Cyber-physical Systems. 15–26, 31–50, 52–54, 57–59, 62, 63, 66–72, 75–77, 83–88, 90–92, 94–101, 103–108, 110–113, 115–117, 120, 122, 124–126, 128, 130, 132–139, 141, 144–147, 156, 158–166, 170, 171, 175, 177, 179, 182, 186, 197, 203, 204, 213, 215–217, 219–235, 237–248, 281–283, 285, 297

CPSoS Cyber-physical Systems of Systems. 36, 37, 161

DSL Domain-specific Language. 49, 54, 108

EAI Enterprise Application Integration. 15, 16, 40

ECA Event–Condition–Action. 26, 48, 49, 72, 85, 108, 109

EMF Eclipse Modeling Framework. 107, 124, 133

EPC Event-driven Process Chain. 26, 50, 84, 229

EPL Event Processing Language. 50, 55, 84, 85, 108, 111, 113, 121, 122, 175, 178, 179, 187, 191, 192, 217–220, 229, 230, 239, 246, 247

IDE Integrated Development Environment. 107, 108, 110, 111, 118, 123, 132, 133, 144, 166, 167, 224

IoT Internet of Things. 15, 16, 18, 21, 30–32, 34, 38, 41, 42, 45, 47–58, 60, 62, 63, 65, 66, 68, 69, 71, 72, 75, 83, 88, 91–93, 96, 101, 111, 116, 117, 121, 124–126, 137–139, 144, 158, 168, 170, 174, 176, 184, 192, 196, 198, 200, 213–216, 218–221, 228, 230, 233, 237–239, 241–244, 246–248, 281, 282

JSON JavaScript Object Notation. 81, 82, 106

KPI Key Performance Indicator. 99, 105, 112, 141, 142, 158, 164, 226

MAPE-K Monitor–Analyse–Plan–Execute over a shared Knowledge. 35, 36, 64, 65, 68–72, 87, 95, 96, 98, 100, 105, 112, 120, 138, 141–155, 157, 159, 160, 162, 164–166, 168, 170, 171, 185, 187–189, 191–197, 201, 202, 207–211, 213, 214, 216, 218, 223–231, 233–235, 238–242, 244–247, 281, 283

MOF Meta-Object Facility. 76, 162

openHAB Open Home Automation Bus. 38, 87, 88, 121, 122, 124, 125, 168, 169, 173–175, 179, 180, 182, 188, 203, 214, 220

OSGi Open Services Gateway initiative. 46, 83, 84, 117, 123, 124, 297

PACPS Process-aware Cyber-physical Systems. 38, 39, 41, 42, 91, 237, 243

PAIS Process-aware Information System. 25

PDCA Plan–Do–Check–Act. 25

QoS Quality of Service. 64, 65, 99, 105, 112, 141, 142, 152, 158, 164, 213, 226

RDF Resource Description Framework. 88

REST Representational State Transfer. 30, 39, 47, 82–84, 98, 111, 122, 124, 125, 131, 137, 146, 149, 165, 168–170, 175, 176, 179, 181–184, 187, 198, 214, 215, 228, 297

ROS Robot Operating System. 39, 83, 89, 173, 174, 198, 203, 291

RPC Remote Procedure Call. 123

SAL Semantic Access Layer. 122–125, 146, 173–176, 179, 181, 182, 216, 220, 231, 234

SLA Service-level Agreement. 64, 65

SLAM Simultaneous Localization and Mapping. 39, 177, 197, 199, 203, 293

SOA Service-oriented Architecture. 15, 28–30, 48, 54, 56, 65, 83, 139, 164, 165, 281

SOAP Simple Object Access Protocol. 39, 46, 47, 83, 111, 169, 170, 228, 297

SoS System of Systems. 31, 95, 126, 223

SPARQL SPARQL Protocol And RDF Query Language. 92, 93, 108, 111, 125, 175, 179, 181, 198, 220, 239, 247, 287

SpEL Spring Expression Language. 97, 145

SQL Structured Query Language. 85

UML Unified Modeling Language. 20, 47, 76, 95

URI Uniform Resource Identifier. 83, 88, 122, 125, 126, 129, 151, 157, 220, 222, 223

WADL Web Application Description Language. 29, 83

WAMP Web Application Messaging Protocol. 123, 132

WfMS Workflow Management System. 16–20, 23–25, 28, 30, 33, 35, 39, 41–49, 53, 56, 58, 62, 67–69, 71, 72, 77, 87, 101, 105, 108, 111–113, 115–118, 125–129, 132, 137–139, 141, 142, 145, 147, 155, 158, 164–168, 170, 171, 173, 174, 176–178, 181, 182, 184, 187, 188, 196, 197, 202, 204, 207, 208, 212–217, 219–223, 226–235, 237–243, 246, 248, 282, 285

WS-BPEL Web Services Business Process Execution Language. 26–28, 46, 47, 49, 53, 54, 56–58, 65, 66, 75, 78, 81, 95, 168, 170, 214, 246, 281, 287

WSDL Web Services Description Language. 29, 83, 170

WSN Wireless Sensor Network. 50, 54

XML Extensible Markup Language. 27, 49, 81, 82

XML-RPC Extensible Markup Language Remote Procedure Call. 83

XPDL XML Process Definition Language. 49

YAWL Yet Another Workflow Language. 26, 28, 47, 49, 112, 168, 169, 214, 248, 281, 283

List of Figures

List of Tables

List of Listings

Appendices

A. DROiT API

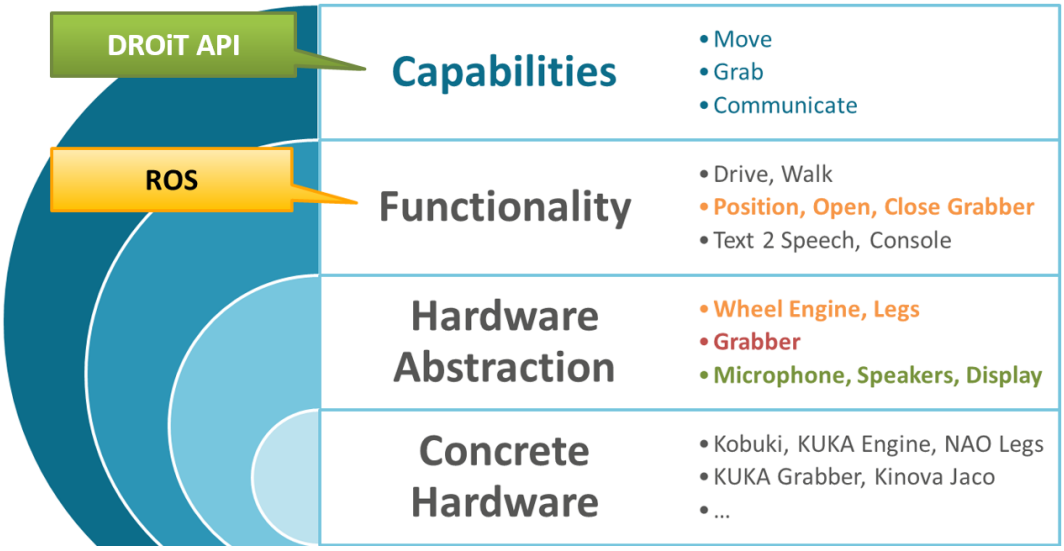


Figure A.1.: Abstraction Levels for Service Robot Platforms [SSAS15].

The high-level programming framework (*DROiT API* [SSAS15]) uses capabilities of robots–namely *Grabbing*, *Movement* and *Communication*–as abstractions for adding the class of service robots as representations of complex sensor–actuator compounds to the set of CPS devices. Compared to using concrete operations (e. g., open/close grabber, drive), this leads to a more abstract view on robot platforms, which hides implementation details and internal processes. Figure A.1 presents an overview of different abstraction levels for domestic service robots ranging from concrete hardware platforms on the lowest level to capability-based abstractions on the highest level. ROS enables robot integration on the level of concrete functionality [QCG+09], whereas the *DROiT API* programming abstraction facilitates integration and robot programming on the capability level [SSAS15]. We modelled the robot's actuating functionalities regarding driving, grabbing and communication based on the actuator properties described in Section 4.3.2 and the abstractions presented in the following sections. The relationship *hasCapability* is used to define a robot's capabilities in the knowledge base (cf. Figure 4.13). The *DROiT API* supports the capabilities of (remote controlled and semi-automatic) *Movement*, (remote controlled and semi-automatic) *Grabbing* of items and two-way-*Communication*. These capabilities were added as nodes to the knowledge base and connected to the specific robot types.

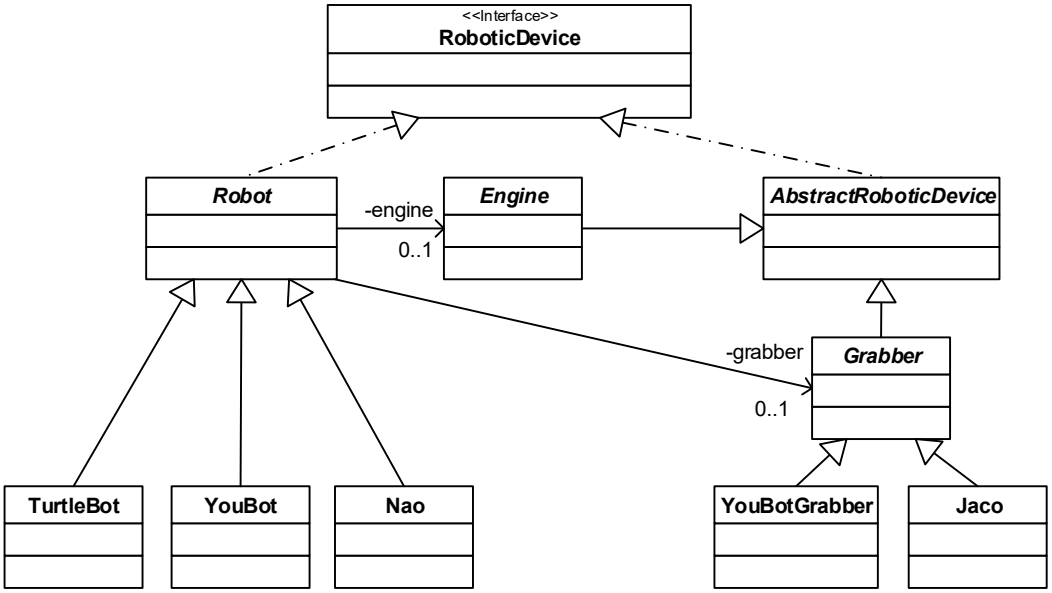


Figure A.2.: Main Abstractions used within the DROiT API for Small Domestic Service Robots.

Figure A.2 shows the core classes of the *DROiT API*. The abstract *Robot* class serves as base class for concrete robot types (e. g., TurtleBot[1], YouBot[2] and Nao[3] robots). A robot's movement capability is enabled by the *Engine* class and its grabbing capability is represented by the *Grabber* class and its concrete subclasses (e. g., the YouBotGrabber or the Jaco[4] arm). Both robot components are viewed as *AbstractRoboticDevice*. Robots and abstract robotic devices implement a common interface (*RoboticDevice*) handling a device's network connection. Figure A.3 presents the robotic platforms supported by the *DROiT API*: a) TurtleBot 2; b) Nao robot; c) Kuka YouBot with Grabber; and d) Jaco arm.

Figure A.3.: Service Robot Platforms Supported by the DROiT API.

[1] http://www.turtlebot.com/turtlebot2/
[2] http://www.youbot-store.com/
[3] https://www.ald.softbankrobotics.com/en/robots/nao
[4] http://www.kinovarobotics.com/assistive-robotics/products/robot-arms/

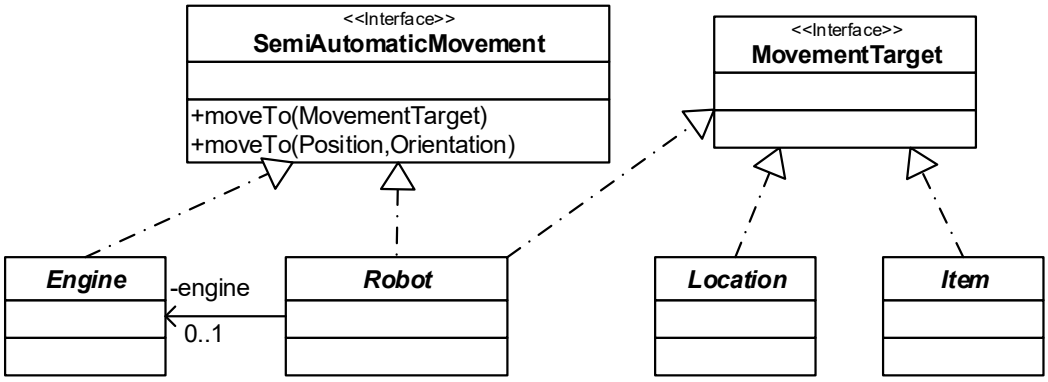


Figure A.4.: Abstractions for the Capability of Semi-automatic Movement.

Movement One of the basic capabilities of a domestic service robot is its ability to move within its environment. The API abstracts two modes of movement: direct control and semi-automatic movement. On the one hand, it is possible to use the movement capability to directly control the robot, moving it to various directions at various speed levels, e. g., via a remote gamepad or keyboard sending control commands to the robot. Many domestic robots offer different means of basic or more advanced SLAM functionalities to navigate within their environment (e. g., using camera-based processes). Therefore the capability to move semi-automatically to given destinations within the robot's world model is added to the knowledge base. Using internal path and obstacle avoidance processes, the robot moves to a provided target on a map (e. g., selected on a user interface within our mobile *SmartCPS* app [SSAS15]). Figure A.4 depicts details regarding this capability. *Robots* or their *Engines* have to implement the *SemiAutomaticMovement* interface depending on whether the robot has a dedicated locomotion engine or built-in movement functionality. The robot has to be provided with a position and orientation–based on coordinates of its internal map–or with a specific *MovementTarget*, which can be a location, item or robot.

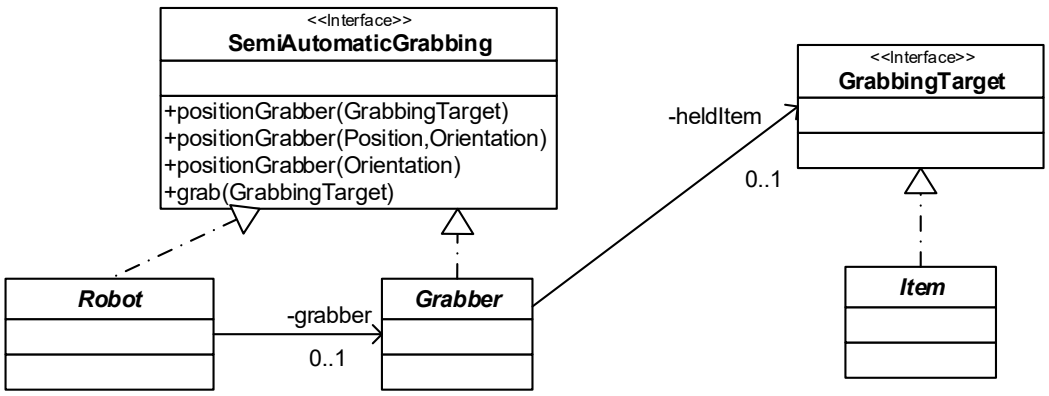


Figure A.5.: Abstractions for the Capability of Semi-automatic Grabbing.

Grabbing The capability of *Grabbing* can be enabled by robots equipped with a grabbing unit or standalone robotic grabbers. Direct remote-controlled grabbing uses the movement and rotation of a virtual point in the palm of the grabbing unit and inverse kinematics as basic control mechanism [GBF85]. This level of abstraction makes technical details such as the degrees of freedom of the grabber transparent to programmers using the API. Analogous to the semi-automatic movement capability, the grabbing of items can also be performed semi-automatically. The *SemiAutomaticGrabbing* interface defines operations a robot or a grabber has to implement in order to grab a target item based on its position and orientation (cf. Figure A.5).

Communication Basic input/output communication in the form of speech synthesis/recognition or textual I/O to communicate with users is also provided by the *DROiT API*. Based on the way the robot is able to handle input/output (e. g., via microphone/speakers for the Nao robot or keyboard/screen for the TurtleBot robot), textual or acoustic communication channels can be used. Figure A.6 shows the abstraction of different communication methods. Basic I/O capability for communicating with users is achieved by the *RobotIO* class providing abstractions for console and audio input/output. Textual I/O is realized by the *ConsoleRobotIO* class for robots that possess a display and means for textual input. Audio I/O is implemented by means of the *AudioRobotIO* class relying on speakers and microphones.

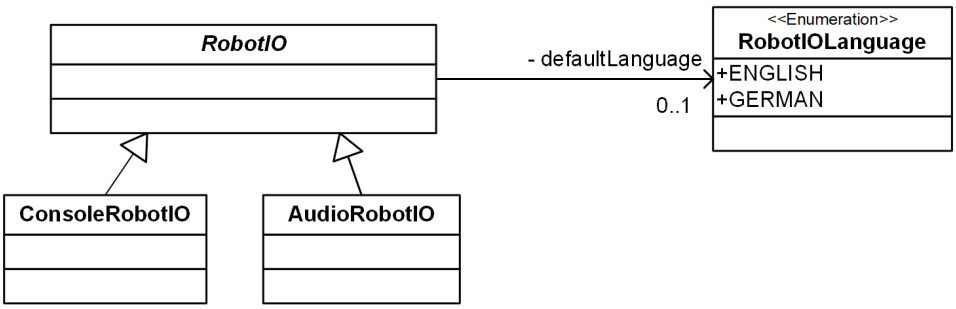


Figure A.6.: Abstractions for the Capability of Communication.

B. Exemplary Consistency Style Sheet

Listing B.1: Exemplary Consistency Style Sheet with Multiple Goals for the Extended Morning Routine Scenario Process.

```
Process:"MorningRoutineExtended"

"SwitchOnLight" : {
 "name": "enough light for working",
 "objectives": [
  { "name": "kitchen light > 700 lux in 5 seconds",
    "satisfiedCondition": "#lightIntensity > 700",
    "compensationCondition": "#objective.created.isBefore
       (#now.minusSeconds(5))",
    "consistencyLevel":0.9,
    "contextPaths": [
     "MATCH (thing)-[:type]->(sensor {name: 'LightSensor
        '})",
      "MATCH (thing)-[:isIn]->(room {name: '
         Kitchen_Mueller'})",
      "MATCH (thing)-[:hasState]->(state:
         LightIntensityState)",
      "MATCH (state)-[:hasStateValue]->(value)",
      "WHERE toFloat(value.realStateValue) > 0",
      "RETURN avg(toFloat(value.realStateValue)) AS
         lightIntensity" ]
  }
 ]
}

"MakeCoffee" : {
 "name":"Coffee is ready",
 "objectives":[
 { "name":"coffee temperature > 37 degrees in 3 mins",
   "satisfiedCondition":"#coffeeTemp > 37",
   "compensationCondition":"#objective.created.isBefore(#
      now.minusSeconds(180))",
   "contextPaths":[
    "MATCH (ctemp {name: '
       State_tinkerforge_irTemp_irTemp_1'})-[:
       hasStateValue]->(value)",
    "WHERE toFloat(value.realStateValue) > 0",
    "RETURN toFloat(value.realStateValue) AS coffeeTemp"
       ]
  }
 ]
}

```

```
"MoveRobotToPaperBoy" : {
 "name":"Robot Position",
 "objectives":[
 { "name":"robot reached the desired position",
   "satisfiedCondition":"#robotReachedPosition(#position,
       5.8D , 13.0D , 1.0D) == true",
   "compensationCondition":"#movement.contains('ARRIVED')
      ",
   "contextPaths":[
    "MATCH(posNode {name: '
       State_proteus_turtle_simplePosition_1'})",
     "MATCH(posNode)-[:hasStateValue]-(posValue)",
     "MATCH(moveNode {name: '
        State_proteus_turtle_movement_1'})",
     "MATCH(moveNode)-[:hasStateValue]-(moveValue)",
     "RETURN posValue.realStateValue as position,
        moveValue.realStateValue as movement" ]
  }
 ]
}

"RetrievePaper": {
 "name":"robot has sufficient battery",
 "objectives" : [
 { "name":"battery level > 30%"
   "satisfiedCondition": "#batteryValue > 0.6,
   "compensationCondition": "#batteryValue < 0.3"
   "contextPaths": [
        "MATCH(n:NeoProcess {processId:'RetrievePaper'})
           -[r:RUNS_ON]->(p:NeoPeer)-[hm:HAS_METRIC]->(m
           :NeoPeerMetric)",
        "RETURN toFloat(m.batteryValue) AS batteryValue"
           ]
  }
 ]
}

"MoveRobotToReader": {
 "name":"execution conformance and liveliness",
 "objectives" : [
 { "name":"heartbeat < 5 seconds and executed",
   "satisfiedCondition": "#processState == 'executed'",
   "compensationCondition": "#timeFrom(#heartBeat).
      isBefore(#now.minusSeconds(5)) and #processState
      == 'executing'"
   "contextPaths": [
    "MATCH(n:NeoProcess {processId:'MoveRobotToReader'})
       -[r:RUNS_ON]->(p:NeoPeer)",
     "RETURN n.state AS processState, p.lastHeartbeat
        AS heartBeat" ]
  }
 ]
}
```

C. Complete Process Metamodel

Figure C.1 shows the complete metamodel developed in the course of this thesis to model CPS workflows. Following, we provide an overview of important metaclasses:

- **ProcessStep**: The central abstract class representing a process step.
- **Process**: A closed self-contained process ready to be executed.
- **AtomicStep**: An activity in a process that cannot be decomposed any further.
- **CompositeStep**: An activity in a process that consists of other process steps.

- **Invoke**: The abstract class representing a service invocation as process step.
- **RESTInvoke**: An atomic process step to execute a call to a REST service.
- **SOAPInvoke**: An atomic process step to execute a call to a SOAP service.
- **OSGIInvoke**: An atomic process step to execute a call to an OSGi service.
- **SemanticAskInvoke**: A dynamic service call to evaluate sensor data.
- **SemanticSelectInvoke**: A dynamic service call to retrieve sensor data.
- **SemanticCommandInvoke**: A dynamic service call to activate actuators.
- **GoalBasedInvoke**: A dynamic service invocation based on a goal definition.

- **HumanTask**: A manual human activity within a process.
- **TriggeredEvent**: An event triggered in the process based on event patterns.
- **TriggeringEvent**: An event produced by the process.
- **LoadClassStep**: An atomic process step to execute custom program code.

- **ControlPort**: Defines ingoing and outgoing activations of a process step.
- **DataPort**: Defines ingoing and outgoing data of a process step.
- **DataType**: Defines the type of the data of a data port.
- **EscalationPort**: Used to define timeouts for process step executions.
- **Transition**: Used to connect ports of succeeding process steps.

- **CpsStep**: Used to define goals for cyber-physical process steps.
- **ManagedStep**: Used to define goals for managed process steps.

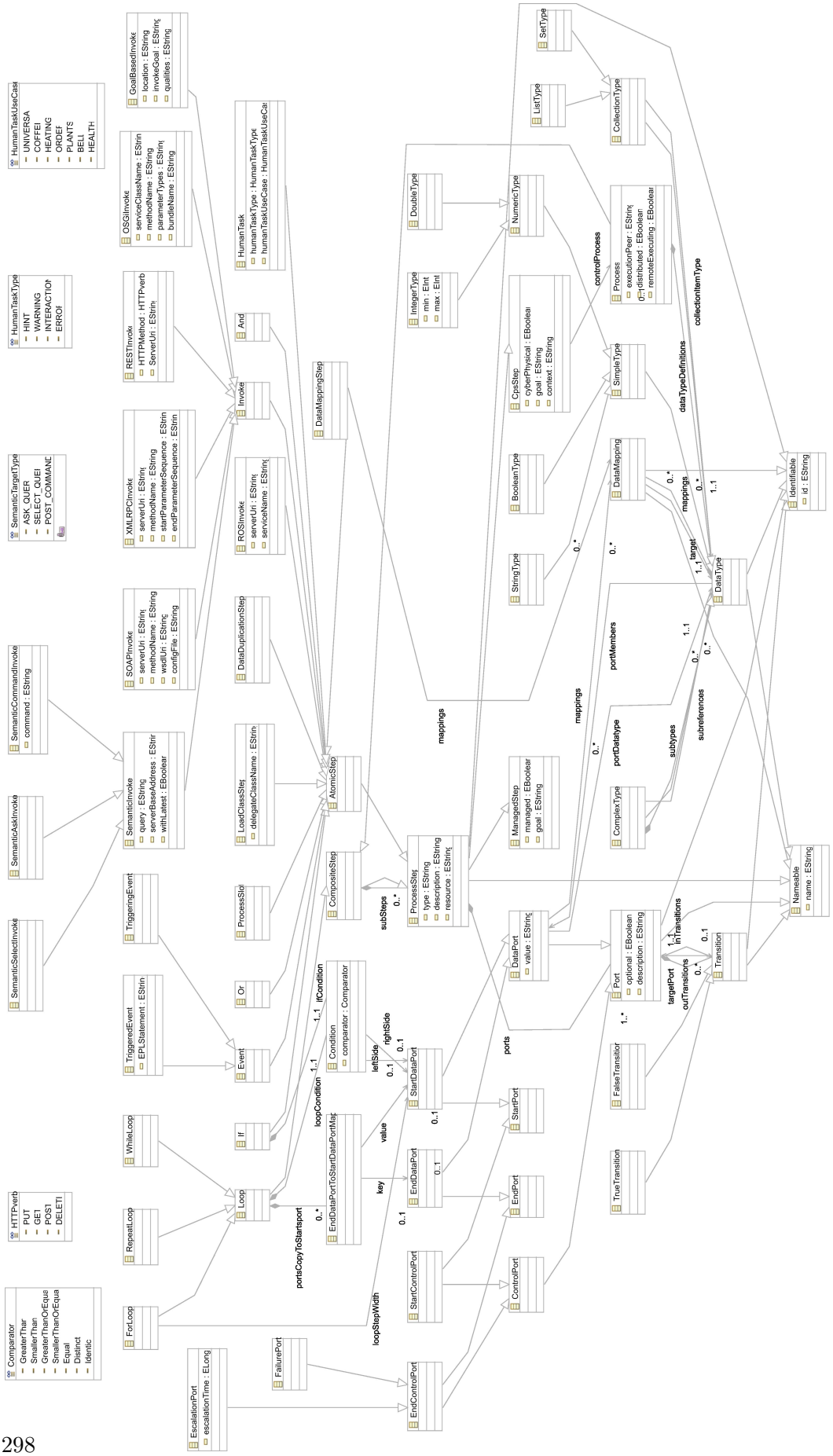


Figure C.1.: Complete Process Metamodel as an Ecore Model.